ELSEVIER Parallel Computing 22 (1996) 1169–1195

PARALLEL COMPUTING

Some aspects about the scalability of scientific applications on parallel architectures

Ignacio Martín Llorente [a,*], Francisco Tirado [a], Luis Vázquez [b]

[a] *Departamento de Informática y Automática, Universidad Complutense, 28040 Madrid, Spain*
[b] *Departamento de Matemática Aplicada, Universidad Complutense, 28040 Madrid, Spain*

Received 25 October 1995; revised 5 June 1996

Abstract

This paper presents the most significant results of the investigation project aimed to implement a parallel application for simulating nonlinear optic systems. A new scalability analysis of parallel algorithm-architecture combinations (parallel systems) is proposed. The execution time and parallel efficiency are the parameters that the parallel system must preserve as it is scaled. These parameters are studied considering the physical phenomena that the scientific application is simulating. The goal in scaling a parallel system is to reduce the overall simulation error in the scientific application so that the simulation reflects the physical phenomena more accurately. Consequently, all the error sources, and not only the problem size, should be scaled simultaneously. The realistic scaling improves the operation count and scalability of common iterative methods to solve systems of equations. The execution time, performance and scalability of a d-dimensional full multigrid algorithm on a d-dimensional mesh of processors are analytically determined. The analytical results are experimentally verified using the parallel simulator of optic systems.

Keywords: Parallel processing for scientific computing; Scalability; Performance evaluation; Computational complexity; Iterative methods

1. Introduction

Numerical simulations are necessary in science when the mathematical model representing a physical phenomena cannot be solved exactly. Often these mathematical models are described by means of nonlinear partial differential equations. Numerical

[*] Corresponding author. Email: llorente@eucmax.sim.ucm.es.

solutions of these equations are obtained converting the continuous problem into a discrete problem. The problem becomes one of solving sparse systems of equations via a discretization of the partial differential equations. Resolution of these systems can exceed the capabilities of even the most powerful conventional computers, and their implementation on parallel computers is thus a natural consideration. In fact, solving these systems of equations is one of the main goals of scientific parallel computing.

When we apply direct methods to these large sparse systems of equations, fill-in occurs and increases the operation count and memory requirements. Moreover, it destroys the locality of the communications, given by a discretization method, producing low parallel efficiencies. The alternative is to use iterative methods. The optimal complexity of multigrid iterative methods – theoretically they can obtain the solution to truncation error accuracy in time proportional to the number of unknowns – together with the fact that their components are highly parallel, makes it natural to study the numerical and parallel properties of these methods.

Several metrics for evaluating the scalability of an algorithm-architecture combination (parallel system) have been proposed in the literature [8,10,19], where the scalability is usually defined as the capability of a parallel algorithm to make effective use of an increasing number of processors in a parallel architecture. It is our opinion that most of these performance-based metrics, because of their failure to consider execution time, are of only theoretical interest.

Execution time and parallel efficiency are the parameters that the parallel system must preserve as it is scaled. By this we mean that the numerical and parallel properties of the system must not deteriorate as the number of processors increases [11]. The proposed scalability analysis considers the next three parameters in the system: efficiency, execution time, and memory usage or final accuracy. An algorithm-architecture combination is perfectly scalable when increasing linearly the problem size with the number of processors, but we obtain a more accurate solution with the same efficiency in the same execution time.

On the other hand, most of the metrics proposed in the literature do not consider the physics problem that the parallel algorithm is solving. The scalability of parallel systems is studied without considering the applications that use them. Usually, the problem size grows with the number of processors to obtain a more accurate solution simulating a physical system. The error due to the spatial discretization is reduced increasing the data set size. However, the global error is not reduced if it has other error sources. Clearly, we have to scale simultaneously all the error sources to achieve a more accurate solution, which is called realistic scaling [17]. It is important to scale the scientific applications in a realistic way, if not, we are performing a larger simulation to obtain a solution with the same accuracy.

Little work has been done before on the effect of reducing all parameters governing error sources, realistic scaling, on the computational complexity of real applications. As we show in this paper, the realistic scaling modifies the computational complexity of some iterative algorithms. The number of Jacobi iterations and multigrid cycles, required for converging to the level of truncation, remains constant in the time dependent case. They therefore can converge a solution to truncation error accuracy in time proportional to the grid size, which is optimal complexity.

This paper is organized as follows. Section 2 describes the process to obtain numerical solutions of the two-dimensional Nonlinear Schrödinger (NLS) and Maxwell–Bloch (MB) partial differential equations using a full multigrid method on a topological mesh of processors. A new scalability analysis is briefly described in Section 3. Relations of the new scalability metrics with other metrics proposed in the literature are presented in the same section. The effect of realistic scaling on the computational complexity of some iterative methods is presented in Section 4. Finally, the scalability of the d-dimensional full multigrid method on a d-dimensional mesh of processors is studied in Section 5. First, the execution time, efficiency and scalability are approximated using a deterministic analytical model and, then, these expressions are experimentally verified in the two-dimensional case with the parallel system described in Section 2.

2. Numerical solution process

In this section we describe briefly a process to obtain numerical solutions of mathematical models from nonlinear optics using parallel multigrid techniques. Fig. 1 shows the steps of the general process to obtain numerical solutions on parallel computers. Though it is difficult to delimit and to define exactly each step, we will give a brief general description and a discussion of the NLS and MB equations.

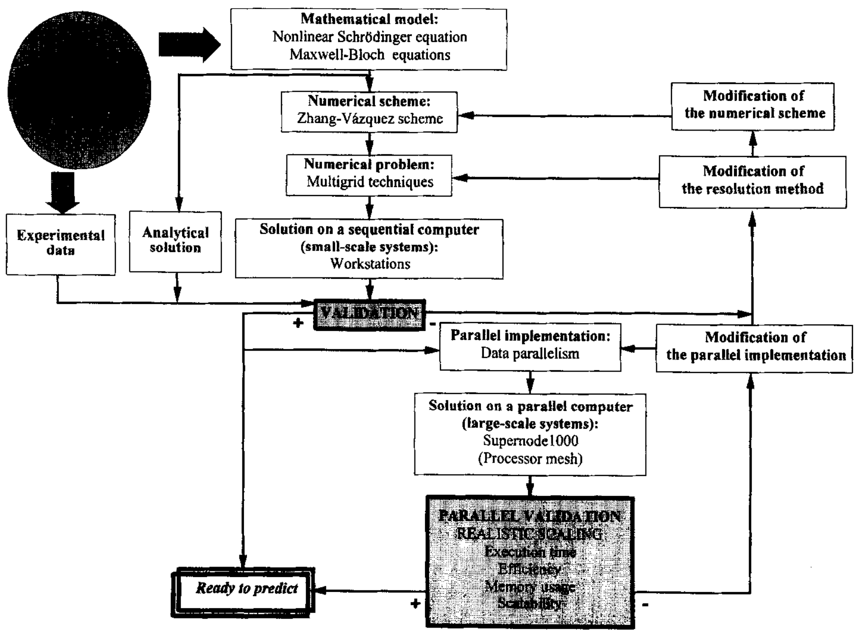


Fig. 1. Solution process of mathematical models on parallel computers.

The formulation of a mathematical model by way of the balance of forces or other factors is the first step to simulate a physical system. Usually, these mathematical models are systems of partial or ordinary differential equations. In our case, these continuous equations are described in Section 2.1.

The space or space-time continuum is replaced by a space or space-time mesh, and difference equations involving algebraic relationships between grid points are obtained via a discretization of the partial differential equations. Finite difference schemes are a natural technique for partial differential equations because of their ease of use and the direct physical interpretation they provide (also because of their flexibility when symmetry or boundary conditions are to be changed). We applied one of these numerical methods to the NLS and MB equations: the Zhang–Vázquez scheme [6]. Some features of this implicit numerical scheme are explained in Section 2.2. When an implicit numerical method is applied to a time dependent partial differential equation, the problem becomes one of solving a sparse system of equations at each temporal step.

If we want to obtain accurate solutions, we must use fine grids. Fine grids produce very large systems of equations, with the number of equations depending on the number of points in the grid, and the shape of the system matrix depending on the number of space dimensions and on the numerical scheme used. In our case, it is tridiagonal for the one-dimensional problem and pentadiagonal or block-tridiagonal for the two-dimensional version. The choice of fast solvers for these systems in general depends on the structure of the problem and on the type of computer being used [16]. We have chosen the multigrid method [2,3,15,21] because its computational complexity approaches the minimum count of $O(N^d)$ operations, where d is the number of spatial dimensions and N^d is the number of points in the grid and unknowns in the system. Some features of the multigrid techniques are explained in Section 2.3.

Once the scientific application produces numerical solutions, it is necessary to validate the previous steps. For this purpose, we propose the consideration of three factors:

- *Verification*: Compare the numerical solutions with an analytical solution for some initial and boundary values. This can be done if the mathematical model can be exactly solved for some special cases: one-dimensional case, without special terms, etc. For example, the one-dimensional NLS equation is tractable analytically, but in general, the perturbations of this equation as driving forces, coupled NLS systems, and the multidimensional equation cannot be solved exactly [6]. On the other hand, if the equation is modeling a physical problem, it is possible to compare the computer results with whatever experimental or observational data is available. This approach was used to validate the Maxwell–Bloch system simulator [12].
- *Complexity*: Achieve the solution with an operation count closer to the optimal one. This depends on the continuous equation, the numerical scheme and the equation system solver.
- *Numerical errors*: Discretization or truncation error (due to the approximation of continuous problems by discrete ones), rounding errors (computers work with a finite number of digits) and convergence errors (the iterative methods obtain the solution when a convergence condition is satisfied).

Now, we have the scientific application ready to be used for prediction. However, resolution of large systems can exceed the capabilities of even the most powerful conventional computers, and the application has to be implemented on a parallel computer. The parallel implementation depends on the target parallel architecture (topology, computation-communication ratio, etc.). Data parallelism model is often used to simulate scientific problems. Section 2.4 describes the parallel full multigrid method.

The last step in the process is the parallel validation. There are many parameters to consider in a parallel system. First, we have to study its numerical and parallel properties (execution time and efficiency) [10]. If these properties are not as good as expected, we have to change the data distribution to avoid processor idleness or to decrease the number of communications. Second, we have to study the scalability of the parallel system using realistic scaling. The numerical and parallel properties of the system must not deteriorate as the system is scaled to obtain more accurate solutions. The scalability of scientific applications is studied in Section 3 and Section 4, and the execution time, efficiency and scalability of parallel multigrid methods are studied in Section 5.

2.1. Mathematical models

2.1.1. The nonlinear Schrödinger equation

The NLS equation arises as an asymptotic limit of a slowly varying dispersive wave envelope in nonlinear mediums such as nonlinear optics, water waves, plasma physics, biomolecular dynamics, etc. [6]. The two-dimensional Eq. (1) is not tractable analytically. Therefore, numerical simulations are needed, for example, to study the propagation of optical solitons in optics fibers. Optical solitons are currently under study since they are expected to provide ultra-high speed telecommunications [9].

$$i\frac{\partial W}{\partial t}+\frac{\partial^2 W}{\partial x^2}+\frac{\partial^2 W}{\partial y^2}+a|W|^2W=0. \tag{1}$$

The physical field $W(t,x,y)$ is a complex function and a is a real constant. The problem is to describe the evolution of this field given an initial condition.

2.1.2. The Maxwell–Bloch equations

Laser devices have become so widely used that the understanding of its behaviour is a question of highly practical interest. However, the conventional theoretical approach is limited and cannot explain the complex features appearing in some very disordered dynamic states that appear under the increase of either the excitation of the medium (the gain) or the transverse dimensions of the laser (aspect ratio). For this purpose, and for studying the effects of symmetry changes on the dynamics it is necessary to simulate the complete laser equations.

The simplest laser model that takes the transverse effects into account is a set of partial differential equations called the Maxwell–Bloch equations:

$$-i\frac{1}{4\beta}\Delta_\perp F+\frac{\partial F}{\partial s}+\sigma(F-P)=0, \tag{2}$$

$$\frac{\partial P}{\partial s} = -(1+i\Delta)P + FD, \tag{3}$$

$$\frac{\partial D}{\partial s} = -\gamma\left[D - r + \frac{1}{2}(F^*P + FP^*)\right]. \tag{4}$$

Energy enters the system though the pumping terms (r) and dissipates through the loss terms (σ). The physical fields are $F(t,x,y)$ and $P(t,x,y)$ (complex functions), and $D(t,x,y)$ (real function); Δ, γ and β are real constants. The problem is to describe the evolution of these fields, which are related to the intensity of the laser beam at each spatial point (x,y), given an initial condition.

2.2. Numerical scheme: The Zhang–Vâzquez scheme

Several numerical schemes have been proposed for the NLS equation [20]. However, we have developed the first conservative scheme that is globally linearly implicit. This is the Zhang–Vâzquez scheme [6]. Let us define an $N+1xN+1$ discrete reticulum (N is the number of spatial steps per side in the grid) with spatial and temporal stepsizes h and τ, and the discrete field: $W_{ij}^n = W(n\tau, ih, jh)$. The discretization of Eq. (1) results in the following system of equations, for $i,j = 1,\dots,N-1$:

$$\begin{aligned} i\frac{W_{ij}^{n+1} - W_{ij}^{n-1}}{2\tau} = &-\frac{1}{2h^2}\left(W_{i+1j}^{n+1} + W_{i-1j}^{n+1} + W_{ij+1}^{n+1} + W_{ij-1}^{n+1} - 4W_{ij}^{n+1}\right)\\ &-\frac{1}{2h^2}\left(W_{i+1j}^{n-1} + W_{i-1j}^{n-1} + W_{ij+1}^{n-1} + W_{ij-1}^{n-1} - 4W_{ij}^{n-1}\right)\\ &-\frac{a}{2}|W_{ij}^n|^2\left(W_{ij}^{n+1} + W_{ij}^{n-1}\right). \end{aligned} \tag{5}$$

The existence of conserved quantities is a good property that is usually related to long time accuracy, while the linearity is essential when applying fast solvers like the linear multigrid algorithm to be presented later. This numerical scheme is second-order accurate in space and first-order accurate in time. Since it is a three-level scheme, a Crank–Nicholson-type scheme is used to start the simulation because the initial data provide only the first temporal step.

Reordering Eq. (5) to get the matrix form, it can be rewritten:

$$\begin{aligned} &W_{i+1j}^{n+1} + W_{i-1j}^{n+1} - \left(4 - ah^2|W_{ij}^n|^2 - i\frac{h^2}{\tau}\right)W_{ij}^{n+1} + W_{ij+1}^{n+1} + W_{ij-1}^{n+1}\\ &\quad = -W_{i+1j}^{n-1} - W_{i-1j}^{n-1} + \left(4 - ah^2|W_{ij}^n|^2 + i\frac{h^2}{\tau}\right)W_{ij}^{n-1} - W_{ij+1}^{n-1} - W_{ij-1}^{n-1}. \end{aligned} \tag{6}$$

As we can see in Eq. (6), we have to solve a pentadiagonal system of equations, $Av = f$, at each temporal step to obtain W_{ij}^{n+1}, $i,j = 1,\dots,N-1$. The system matrix, A, and the right-hand side, f, depend on the function values in the two previous steps n and $n-1$ (three-level scheme, Fig. 2a). On the other hand, a Crank–Nicholson-type scheme only needs the function values in the previous temporal step (two-level scheme, Fig. 2b).

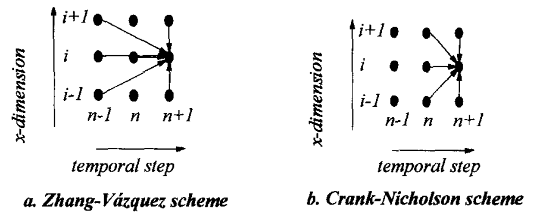


Fig. 2. Zhang–Vázquez and Crank–Nicholson numerical schemes.

The experience obtained applying the Zhang–Vázquez scheme to the NLS equation suggests the application of a similar approach to the problem of the numerical simulation of the MB laser equations. Let us define a $N+1xN+1$ discrete reticulum with spatial and temporal stepsizes h and τ, and the discrete fields $F_{ij}^n \equiv F(n\tau, ih, jh)$, $P_{ij}^n \equiv P(n\tau, ih, jh)$, $D_{ij}^n \equiv D(n\tau, ih, jh)$. The discretization of Eq. (2), Eq. (3) and Eq. (4) produces the following systems of equations, for $i,j = 1,\dots,N-1$:

$$\begin{aligned}
-i\frac{1}{4h^2\beta}\Big(&F_{ij+1}^{n+1} + F_{ij-1}^{n+1} + F_{i+1j}^{n+1} + F_{i-1j}^{n+1} - 4F_{ij}^{n+1} + F_{ij+1}^{n-1} + F_{ij-1}^{n-1} + F_{i+1j}^{n-1} \\
&+ F_{i-1j}^{n-1} - 4F_{ij}^{n-1}\Big) + \frac{F_{ij}^{n+1} - F_{ij}^{n-1}}{\tau} + \sigma\left(F_{ij}^{n+1} + F_{ij}^{n-1} - P_{ij}^{n+1} - P_{ij}^{n-1}\right) = 0,
\end{aligned} \tag{7}$$

$$\frac{P_{ij}^{n+1} - P_{ij}^{n-1}}{\tau} = -(1 + i\Delta)\left(P_{ij}^{n+1} + P_{ij}^{n-1}\right) + \left(F_{ij}^{n+1} + F_{ij}^{n-1}\right)D_{ij}^n, \tag{8}$$

$$\frac{D_{ij}^{n+1} - D_{ij}^{n-1}}{2\tau} = -\gamma\left[\frac{D_{ij}^{n+1} + D_{ij}^{n-1}}{2} - r + \frac{1}{2}\left(F_{ij}^{n*}P_{ij}^n + F_{ij}^n P_{ij}^{n*}\right)\right]. \tag{9}$$

2.3. Numerical problem: Multigrid method

If we want to obtain accurate solutions to the problems presented in Section 2.2, we must use fine grids. Fine grids produce very large systems of equations. Iterative methods are often preferable to direct methods for solving sparse systems. The application of a direct solver to a very sparse system results in fill-in (the introduction of non-zero elements in positions of the matrix that were originally zero). Fill-in increases the operation count and the memory requirements of the direct methods. Concretely, the bandwidth of the matrix of the equation system (6) is equal to N, so the number of flops for a direct method varies as N^4 [16]. Moreover, in a parallel setting, fill-in destroys the inherent locality of the communications given by a finite difference method, resulting in inefficient parallelism.

The most well known of the iterative methods are the relaxation-type methods, such as Jacobi or Gauss–Seidel. Some of these methods, although avoided in practice because they are slow to converge, can be used as part of a more sophisticated multigrid or conjugate gradient methods. Multigrid method can be viewed as a technique for accelerating the convergence of iterative methods. The conjugate gradient method is used to accelerate any iterative method, including multigrid methods. The computational complexity of multigrid methods is closer to optimal than that of conjugate gradient methods. For a comparison of conjugate gradient and multigrid methods see [21]. Specially, only $O(N^2)$ operations are required to solve the system of equations defined in (6).

The problem is that multigrid is a technique, rather than a single method. To apply a multigrid method correctly, it is necessary to find appropriate parameters for the algorithm. If these parameters are not chosen appropriately, the final complexity may be far from optimal. The key idea of the multigrid method can be understood by explaining first the relatively simple case of a two-grid method. The two-grid method is an iterative method: We start with an initial guess, and the final solution is reached after a certain number of two-grid cycles. The next six steps describe a two-grid cycle for the solution of the system (6), $Av^h = f$, over the grid h at a given temporal step:

step 1: Iterate n_1 times using the smoothing iterative method over the original system of equations, reaching an approximation v^h.
step 2: The residual or defect r^h is formed as $r^h = f - Av^h$.
step 3: Prepare to solve the error equation in the coarser grid $2h$. In so doing it is necessary to restrict the residual to that grid $r^{2h} = I_h^{2h} r^h$, where I_h^{2h} is the restriction operator.
step 4: Solve the error equation over the grid $2h$, $A^{2h}e^{2h} = r^{2h}$.
step 5: Apply the prolongation operator to interpolate the correction to the finer grid. $e^h = I_{2h}^h e^{2h}$, where I_{2h}^h is the prolongation operator.
step 6: Add this error e_h to the value calculated in step 1, $v^h = v^h + e^h$.

In the two-grid correction, we have to solve a system of equations at step 4 on $2h$. If this system is solved on a still coarser grid, $4h$, and this process is repeated on successively coarser grids until a Jacobi solution of the residual equation is possible, the algorithm assumes a recursive structure. This recursive algorithm is called MVJ-cycle:

```
$v^h \leftarrow \mathrm{MVJ}^h(v^h, f^h)$:
step 1: If $h$ is the coarsest grid ($N_{\mathrm{lower}} \times N_{\mathrm{lower}}$), solve using $\gamma_1$ Jacobi iterations and return, else continue.
step 2: Relax (smoothing method) $n_1$ times on $A^h v^h = f^h$ with initial guess $v^h$.
step 3: $f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$,
    $v^{2h} \leftarrow 0$,
    $v^{2h} \leftarrow \mathrm{MVJ}^{2h}(v^{2h}, f^{2h})$.
step 4: Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
step 5: Relax (smoothing method) $n_2$ times on $A^h v^h = f^h$.
```

An MVJ-cycle is a V-type multigrid cycle. We start with an initial solution, v^h, and the final solution is reached after a fixed number of MVJ-cycles. The multigrid cycle works on a hierarchy of point grids that represents the physical domain to different

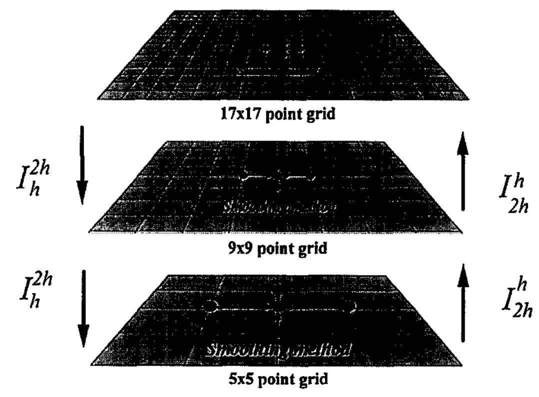


Fig. 3. Multigrid method.

degrees of granularity (Fig. 3). All of them have a number of spatial steps per side power of two. The next coarser point grid has half the number of spatial steps per side that the current grid has.

The smoothing method reduces the high frequency components of the error at each level, and the low components are reduced on coarser levels where they have higher frequencies. This can be done in a recursive way until a level where the system can be solved directly. Restriction and prolongation operators connect the levels. The prolongation operator maps data from a coarser level to the current level, and the restriction operator takes values from the finer level to the current one. It has been proved that the error is reduced by a constant factor at each cycle with an appropriate choice of the multigrid components and parameters. Smoothing, restriction (I_h^{2h}), and prolongation (I_{2h}^h) operators are chosen to achieve good convergence rates and good parallel properties. We have used the damped Jacobi method as the smoothing method, bilinear interpolation as the prolongation operator, and a half-weighting average of neighbouring fine grid values as the restriction operator.

If the Jacobi method is used to solve the system on the coarsest grid, multigrid can be faster and more efficient on a parallel computer. It can be faster because, with time dependent equations, the spectral radius of the Jacobi method matrix depends on N, h and τ parameters (Section 4.2), and then it is possible to get good convergence rates for some values of these parameters. The parallel efficiency increases because the size of the coarsest grid is greater than the size of the processor mesh. When the number of grid points is less than the number of processors, more communication is needed and some processors are idle, resulting in a significant loss of efficiency.

If we define the problem on grids of various sizes, we can obtain the solution at each level, from the coarsest to finer grids, with a fixed number, n_3, of MVJ-cycles, using as initial guess the solution interpolated from the coarser grid. This is the full multigrid algorithm, FMVJ. With this approach, the solution at each level converges to truncation level. The full multigrid process starts with the solution on the coarsest grid, using γ_2 Jacobi iterations.

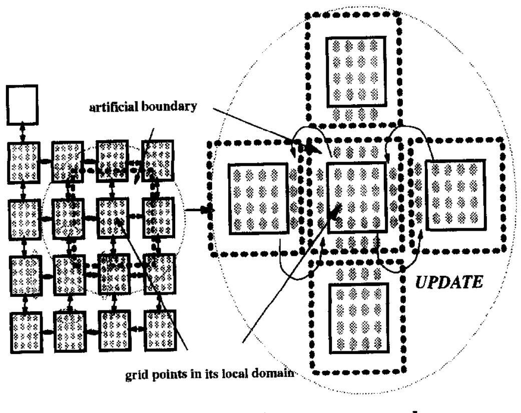
2.4. Parallel full multigrid method

In this section, we describe how to implement the FMVJ method on a distributed-memory parallel computer. An overview of parallel multigrid methods on different architectures, interconnection networks and appropriate smoothing iterations can be found in [4,5,7,11,14].

We use a domain decomposition in physical space for all grids [14]. Each processor runs a copy of the same program and works on a local domain. Data is distributed equally over all the processors at each level. The equations are then distributed over all the processors at each level. Each node contains all the grid points corresponding to its domain, as well as those grid points within an artificial boundary that extends into neighboring domains (Fig. 4). In this way, if the boundary points of a processor are modified, it is necessary to update the copy in the neighbouring processors (artificial boundary).

We next consider how to carry out each step of the MVJ-cycle on a parallel computer. The damped Jacobi iteration is seen to be perfectly parallel (steps 2 and 5 in an MVJ-cycle). Each processor performs the computation over its subset of data (geometric or data parallelism). As each point needs its four neighbouring points at iteration, processors have to communicate their boundary points to the four neighbouring processors after each iteration (Fig. 4). This communication, which updates the artificial boundary of each processor, is done at the same time in all processors. An overview of the implementation of linear-iterative solvers onto a processor array can be found in [1].

Following, we have to calculate the residual (step 3 in an MVJ-cycle). The multiplication $A^h v^h$ can be done in parallel over all processors, since each processor has the updated copy of the artificial boundary points. The edge values of the residual $f^h - A^h v^h$ must be transmitted to update the artificial boundary points of the neighboring processors so that the restriction half-weighting operator can be applied.



16x16 point grid, 4x4 processor mesh

Fig. 4. Artificial boundaries.

Analogous considerations apply at each level until the coarsest level, where the system is solved using γ_1 Jacobi iterations (step 1 in an MVJ-cycle). This is done when the number of grid points is larger than the number of processors.

On the return up (step 4 in an MVJ-cycle) after the correction, the artificial boundaries are updated. Because multigrid is an iterative method, it could require a norm evaluation for the computation of convergence criteria after each MVJ iteration. This is not necessary for some cases because its performance can be predetermined [2].

3. Scalability of parallel systems

Computer scientists are interested in the performance achieved on parallel systems; their primary goal is to obtain good levels of efficiency on larger machines. Therefore, they see scalability as a metric to measure the capability of effectively utilizing an increasing number of processors. They want to design architectures on which algorithms can be implemented efficiently and for this reason base their scalability metrics on performance parameters.

Several performance-based scalability definitions have been presented in literature during recent years. In the same way, some metrics have been proposed to evaluate and compare the scalability of parallel algorithm-architecture combinations. The isoefficiency [8,10] and the isospeed [18,19] concepts are samples of this kind of metrics.

The isoefficiency function, $f(p)$, measures the way the problem size must grow with the number of processors to maintain a fixed efficiency. The isoefficiency function is used to measure the scalability, the more linear $f(p)$ is, the more scalable the parallel system is. Constant efficiency means that speedup increases linearly with the problem size.

The isospeed measures the ability to maintain the average unit speed, where the average unit speed is the speed of the parallel system divided by the number of processors. If the sequential speed is independent of the problem size, the isospeed approach is the same as the isoefficiency approach. The parallel system is scalable if the average speed can be maintained by increasing the problem size as the number of processors grows. The variation of this problem size provides the scalability metric. Let W be the amount of work of an algorithm when p processors are employed, and W' the amount of work needed to maintain the average speed when $p' > p$ processors are employed. The scalability from system size p to system size p' is defined as:

$$\Psi(p,p') = \frac{p'W}{pW'}. \tag{10}$$

It is our opinion that performance-based metrics, because of their failure to consider execution time, are of only theoretical interest. However, some aspects of the relation between the isospeed scalability and execution time have been recently studied in [18].

Computational scientists, on the other hand, are more interested in increasing the accuracy of the solutions to their simulation programs than in performance parameters. Their primary goal is to perform more accurate simulations while maintaining acceptable execution times.

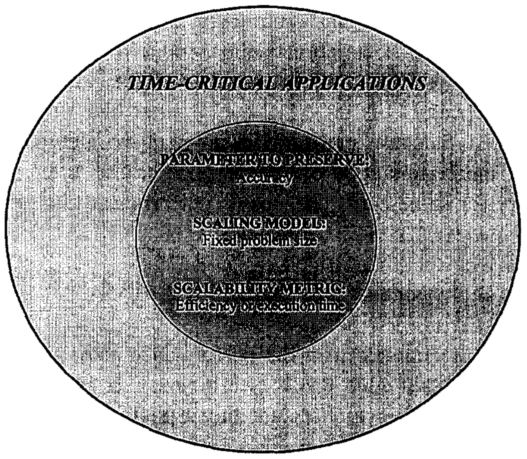


Fig. 5. Time-critical scaling model.

We propose a new scalability analysis in which the numerical and parallel properties are considered as the system is scaled for obtaining more accurate solutions. In running scientific applications, we consider only two ways to scale a parallel system: time-critical and accuracy-critical scaling models.

3.1. Time-critical scaling model

Using the time-critical scaling model (Fig. 5), we are interested in obtaining the same solution reducing the execution time. Then, the parallel system is scaled maintaining a fixed problem size. So, scalability is assessed in terms of efficiency, or execution time, as the number of processors increases.

3.2. Accuracy-critical scaling model

Using the accuracy-critical scaling model (Fig. 6), we are interested in obtaining the most accurate solution without exceeding the system's memory limits. In this case, a scalability analysis must consider the execution time and the efficiency as the parameters that the parallel system must preserve as it is scaled. By this we mean that the numerical and parallel properties of the system must not deteriorate as the number of processors increases.

Our scalability analysis predicts the parallel properties of larger systems based on the properties of smaller systems. We consider that in optimum conditions the parallel system properties must be preserved as the system is scaled. The scalability analysis enables us to scale the problem size with the number of processors to keep a parameter unchanged.

Specifically, the following scaling models correspond to the three parameters for consideration in a parallel system (efficiency, execution time and memory usage or simulation accuracy):

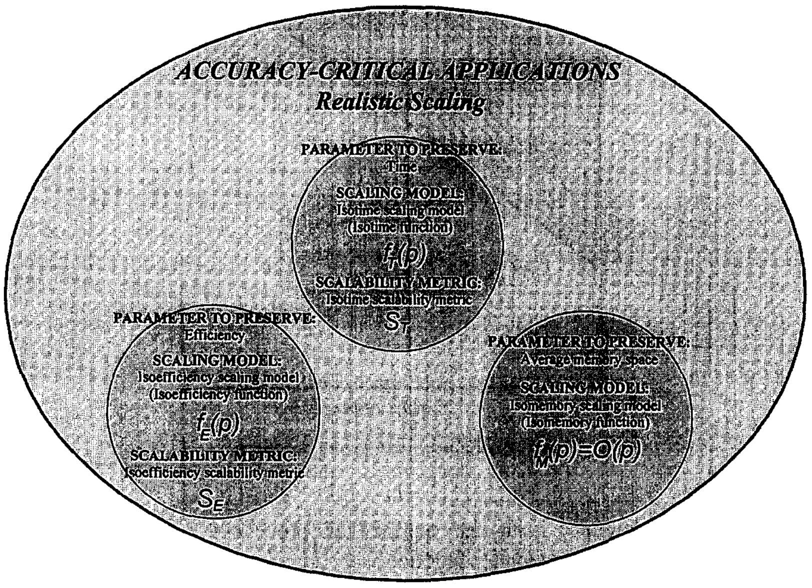


Fig. 6. Accuracy-critical scaling models.

- *Isotime scaling model*: This model is used when we are interested in obtaining the most accurate solution while maintaining the execution time.
- *Isoefficiency scaling model*: This model is used when we are interested in obtaining the most accurate solution while maintaining the parallel efficiency. Consequently, we do not consider the isoefficiency as a scaling metric, but as a scaling model.
- *Isomemory scaling model*: This model is used when we are interested in obtaining the most accurate solution possible, regardless of the efficiency and the execution time.

An isoparameter function describes how to increase the problem size with the number of processors to maintain the parameter. The problem size has to grow linearly with the number of processors to keep the average memory usage unchanged (the isomemory function is linear). In most cases, the problem size has to be increased in a sublinear way to maintain a constant execution time (the isotime function is in the sublinear area), and it has to be increased in a superlinear way to maintain a constant efficiency (the isoefficiency function is in the superlinear area). In fact, the isoefficiency and isotime functions (Fig. 7) of common parallel algorithms are polynomial functions:

- *Isoefficiency functions* $f_E(p)$: They are $O(p^k)$ for some $k \geq 1$, where p is the number of processors. The smaller the k parameter the more scalable is the parallel system with respect to the efficiency.
- *Isotime functions* $f_T(p)$: They are $O(p^k)$ for some $k \leq 1$. The larger the k parameter, the more scalable is the parallel system with respect to the execution time.
- *Isomemory functions* $f_M(p)$: They are $O(p)$.

Using the accuracy-critical scaling model, it is intended to obtain the most accurate solution without exceeding the system's memory limits. The optimum case is then to

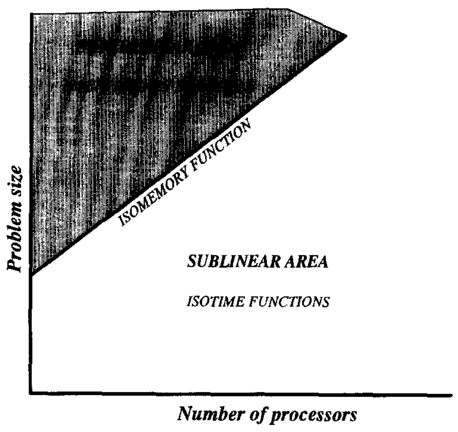


Fig. 7. Isoefficiency and isotime functions.

scale the problem linearly with the number of processors (isomemory function). Therefore, the closer an isoparameter function is to the linear function, the more scalable the parallel system is with respect to the parameter, because the most accurate solution is obtained without exceeding the system's memory limits while maintaining the parameter.

Then, if we are interested in preserving the efficiency, the closer the isoefficiency function is to the isomemory function, the more scalable the parallel system is with respect to the efficiency. Then, the parallel system is perfectly scalable with respect to the efficiency if the isoefficiency function grows linearly. In the same way, the closer the isotime function is to the isomemory function, the more scalable the parallel system is with respect to the execution time. Therefore, the parallel system is perfectly scalable with respect to the execution time if the isotime function grows linearly.

One parallel system is perfectly scalable if it is perfectly scalable with respect to all its parallel parameters; all the isoparameter functions grow linearly. Therefore, they coincide with the isomemory function: If the parallel system is scaled with the isomemory scaling model, the execution time and the efficiency remain constant. This provides a qualitative measurement of the scalability, representing the three isoparameter functions on a graph.

Let us define a quantitative measurement of the system scalability. Perfectly scalable means that all parallel parameters remain constant as the number of processors increases. Consequently, it is possible to define scalability metrics measuring how one parameter deteriorates when the problem size is increased to maintain the other parameter (Fig. 8).

Isoefficiency and isotime scaling models have their own scalability metric. Let N_1 be the problem size when p_1 processors are employed and N_2 the problem size needed to maintain a constant efficiency when $p_2 > p_1$ processors are employed. Therefore, one scalability metric for a parallel system from size p_1 to system size p_2 is to divide the execution time $T(N_1,p_1)$ of the system with problem size N_1 and machine size p_1 by the execution time $T(N_2,p_2)$ of the problem size N_2 using p_2 processors. In this way, it

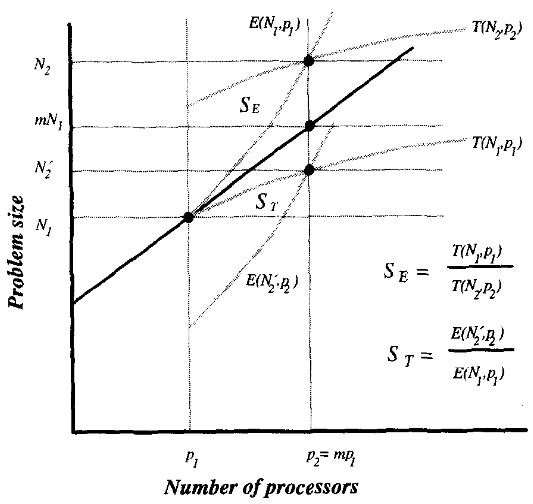


Fig. 8. Isoefficiency and isotime scalability metrics.

is possible to calculate the impact of maintaining constant efficiency in the execution time.

Then, we define the isoefficiency scalability metric S_E from size (N_1,p_1) to size (N_2,p_2), where N_2 is obtained using the isoefficiency function, $N_2 = f_E(p_2/p_1)N_1$ and then $E(N_1,p_1) = E(N_2,p_2)$, as the execution time in the first system size divided by the execution time in the second system size:

$$S_E(p_1,p_2) = \frac{T(N_1,p_1)}{T(N_2,p_2)} = \frac{T(N_1,1)p_2E(N_2,p_2)}{T(N_2,1)p_1E(N_1,p_1)}$$
$$= \frac{T(N_1,1)p_2}{T(N_2,1)p_1} = \frac{T(N_1,1)p_2}{T(f_E(p_2/p_1)N_1,1)p_1} \le 1. \tag{11}$$

This definition is the same as the isospeed scalability metric, $\Psi(p_1,p_2)$ defined in (10), if the sequential execution speed does not vary with the problem size.

In the same way, let us define the isotime scalability metric S_T from size (N_1,p_1) to size (N_2,p_2), where N_2' is obtained using the isotime function, $N_2' = f_T(p_2/p_1)N_1$ and then $T(N_1,p_1) = T(N_2',p_2)$, as the efficiency in the second system size divided by the efficiency in the first system size:

$$S_T(p_1,p_2) = \frac{E(N_2',p_2)}{E(N_1,p_1)} = \frac{(T(N_2',1))/(T(N_2',p_2)p_2)}{(T(N_1,1))/(T(N_1,p_1)p_1)}$$
$$= \frac{T(N_2',1)p_1}{T(N_1,1)p_2} = \frac{T(f_T(p_2/p_1)N_1,1)p_1}{T(N_1,1)p_2} \le 1. \tag{12}$$

Depending on the scaling model used, isotime or isoefficiency, one of both metrics is used to measure the system scalability. If we are interested in maintaining the efficiency,

S_E metric gives the increase of execution time, and if we are interested in maintaining the execution time, S_T metric gives the fall in efficiency. It is necessary to consider both metrics to study the scalability of a parallel system. Multiplication S of both scalability metrics is given by:

$$S(p_1,p_2) = S_T(p_1,p_2)\, S_E(p_1,p_2) = \frac{T(N_2',1)}{T(N_2,1)} = \frac{T(f_T(p_2/p_1),1)}{T(f_E(p_2/p_1),1)} \leq 1. \quad (13)$$

The last expression gives us the relation between the scalability metrics and the isoefficiency and isotime functions, between the quantitative and the qualitative measurements. It can be proved that if one of both metrics is equal to one, the other is also equal to one and the isoefficiency and isotime functions coincide with the isomemory function. Moreover, if the isotime function grows linearly, the isoefficiency function also grows linearly and then the two metrics are equal to one.

We have presented some ideas about scalability of scientific applications on parallel computers. Because the main goal of the accuracy-critical scaling model is to obtain more accurate solutions, the execution time and the efficiency must be studied when the system is scaled in a realistic way. This is critical for the proper evaluation of a parallel system. The scalability analysis must consider the scientific application that it is being scaled.

4. Effect of realistic scaling on the computational complexity of numerical algorithms

The accuracy-critical scaling model is used to scale a scientific application with the number of processors when more accurate solutions are required. The error due to the spatial discretization is reduced increasing the data set size. However, if the simulation global error has other error sources, the global error is not reduced. That is the reason the scaling of only the problem size is called naive scaling [17]. Clearly, we have to scale simultaneously all the error sources to achieve a more accurate simulation. As we show in this section, the realistic scaling modifies the computational cost of some common iterative algorithms. We must not think about algorithm complexity in terms of input data set size but in terms of discretization parameters.

The matrix of the system of equations that arises in the finite-difference representation to a stationary partial differential equation only depends on the spatial stepsize, h. Therefore, convergence rate and computational cost of iterative methods depend on h or N (N is inversely proportional to h). For example, the number of Jacobi iterations for converging the finite-difference representation of the d-dimensional Poisson problem [3] to the level of truncation is $O(N^2 \log N)$. Since, each iteration costs $O(N^d)$ arithmetic operations, then the computational cost is $O(N^{d+2} \log N)$. Here, the truncation error is only affected by the spatial stepsize, and when increasing the problem size a more accurate solution is obtained.

However, the matrices of the systems of equations that arise in the finite-difference representation to a time dependent partial differential equation depend on the spatial and temporal stepsizes, h and τ. Consequently, convergence rate and computational cost of iterative methods depend on h and τ. Both discretization parameters contribute to the global error. The decrease of these parameters, so that their error contributions are equal, maintains the number of Jacobi iterations and multigrid cycles required to solve the problem to the level of truncation. Their computational costs vary as $O(N^d)$, which is optimal order.

In this section, we illustrate, analytically and numerically, this feature using the one-dimensional NLS equation. We consider the one-dimensional case because it can be solved exactly and then it is possible to estimate the truncation error. The ideas and results extend directly to the higher-dimensional cases. The Zhang–Vázquez discretization of the one-dimensional NLS equation produces the following tridiagonal system of equations at each temporal step, $n+1$, for $i=1,\dots,N-1$:

$$\begin{aligned} i\frac{W_i^{n+1}-W_i^{n-1}}{2\tau} &= -\frac{1}{2h^2}\left(W_{i+1}^{n+1}-2W_i^{n+1}+W_{i-1}^{n+1}+W_{i+1}^{n-1}-2W_i^{n-1}+W_{i-1}^{n-1}\right)\\ &\quad -\frac{a}{2}|W_i^n|^2\left(W_i^{n+1}+W_i^{n-1}\right). \end{aligned} \tag{14}$$

4.1. Convergence rate of iterative methods

The following equation represents a linear stationary iterative method of the first degree:

$$v^{(k+1)}=Hv^{(k)}+d. \tag{15}$$

Its asymptotic rate of convergence is determined by the spectral radius $\rho(H)$, where H is the matrix of the iterative method. The spectral radius of a matrix is the largest eigenvalue in absolute value. The error relation of the iterative method is $e^{(k+1)}=He^{(k)}$, so the error after k iterations is $e^{(k)}=H^k e^{(0)}$. This expression is approximately the same as $e^{(k)}=\rho(H)^k e^{(0)}$. Therefore, the number of iterations, k, for converging the solution is given by:

$$\rho(H)^k=\frac{e^{(k)}}{e^{(0)}}\Rightarrow k=\frac{\log(e^{(k)}/e^{(0)})}{\log(\rho(H))}. \tag{16}$$

The number of iterations of the iterative method depends on its convergence rate, $\log(\rho(H))$, the error in the initial guess, $e^{(0)}$, and the error in the final solution, $e^{(k)}$. If we want to converge the solution to the level of truncation, the error must be reduced from $e^{(0)}$ to $e^{(k)}\approx O(h^2)$. The initial guess is the solution in the previous temporal step: $e^{(0)}\approx O(\tau)$. Therefore, the number of iterations, k, for converging to the level of truncation in our case is:

$$k=\frac{\log(h^2/\tau)}{\log(\rho(H))}. \tag{17}$$

4.2. Convergence rate of the Jacobi method

The spectral radius of the matrix of the Jacobi method applied to the system of equations defined in (14) is given by:

$$\rho(H) \approx \operatorname{mod}\left(\frac{2\cos(\pi/(N+1))}{(i(h^2/\tau))-2}\right). \tag{18}$$

The larger the spectral radius is, the slower the convergence rate is. Jacobi method can be fast with time dependent equations because the spectral radius (18) depends on N, h and τ parameters, and then it is possible to get good convergence rates for some values of these parameters. This has been shown experimentally in previous papers [13], where the Jacobi method is compared with other iterative and direct methods. The number of iterations is obtained from Eq. (17) and Eq. (18):

$$\begin{aligned} k &\approx \frac{\log(h^2/\tau)}{\log\left((2\cos(\pi/N))/\left(\sqrt{4+(h^4/\tau^2)}\right)\right)} \\ &= \frac{\log(h^2/\tau)}{\log(2\cos(\pi/N))+\log\left(2/\left(\sqrt{4+(h^4/\tau^2)}\right)\right)} \\ &\approx \frac{\log(h^2/\tau)}{\log\left(1-(\pi^2/N^2)\right)+\log\left(2/\left(\sqrt{4+(h^4/\tau^2)}\right)\right)} \\ &\approx \frac{N^2\log(h^2/\tau)}{\pi^2+N^2\log\left(2/\left(\sqrt{4+(h^4/\tau^2)}\right)\right)}. \end{aligned} \tag{19}$$

Both error sources (h and τ) have to be scaled simultaneously to obtain a more accurate solution. In our case, the error varies as $O(h^2)+O(\tau)$, so if the spatial stepsize is divided by 2, the time stepsize must be divided by 4. Consequently, h^2/τ ratio remains constant and Eq. (19) can be rewritten as:

$$k \approx \frac{N^2A}{N^2B+C}. \tag{20}$$

Usually, C is lower than N^2B and then the number of iterations tends to A/B. This quotient depends on the h^2/τ factor. So, if the system is increased using realistic

Table 1
Spectral radiuses of matrices of the Jacobi method applied to some time dependent partial differential equation (PDE) and numerical scheme combinations

PDE (numerical scheme)	$\rho(H)$
1D NLS equation (Zhang–Vazquez)	$\approx (2\cos(\pi/(N+1)))/\left(\sqrt{(4+(h^4/\tau^2))}\right)$
1D NLS equation (Crank–Nicholson)	$(\cos(\pi/(N+1)))/\left(\sqrt{(1+(h^4/\tau^2))}\right)$
1D Heat equation (Full implicit)	$(2\cos(\pi/(N+1)))/((2+(h^2/\tau)))$
1D Heat equation (Crank–Nicholson)	$(\cos(\pi/(N+1))/((1+(h^2/\tau)))$

scaling, the number of Jacobi iterations for converging the solution to the level of truncation remains constant and depends on the h^2/τ ratio. Same results are obtained in the multidimensional case. The dependence of the convergence rate on the h^2/τ ratio not only appears with the NLS equation and the Zhang–Vázquez numerical scheme. The spectral radiuses of other time dependent partial differential equation and numerical scheme combinations are shown in Table 1.

4.3. Convergence rate of the multigrid method

If the multigrid components are well adjusted, the convergence rate of the multigrid method is independent of the spatial and temporal stepsizes. Then, its spectral radius can be considered as constant. For this reason, the number of cycles (17) for converging the solution to the level of truncation remains constant using realistic scaling:

$$k \approx \frac{\log(h^2/\tau)}{\log(\rho(H))} = \text{constant}. \tag{21}$$

4.4. Numerical results

The analytical results are experimentally verified using numerical simulations. The systems of equations defined in (14), one per temporal step, were solved using Jacobi iterations and multigrid cycles. The one-dimensional NLS equation has an analytical solution given by:

$$W(t,x) = 2\eta e^{i(2\chi x - 4(\chi^2 - \eta^2)t)} \operatorname{sech}(2\eta x - 8\chi\eta t - x_o), \tag{22}$$

where η, χ and x_0 are real constants. Consequently, it is possible to calculate the truncation error of the numerical solution.

The number of iterations and multigrid cycles ($n_1 = n_2 = 3$), for converging the solution to truncation error accuracy at each temporal step, and the truncation error, calculated in the final instant $t = 0.04$, with different spatial and temporal stepsizes are presented in Table 2.$W(0,x)$ was used as solution in the first temporal step. Because Zhang–Vázquez scheme is not a self-starting numerical scheme, a Crank–Nicholson-type scheme was used to perform the second temporal step.

Table 2
Jacobi iterations and multigrid cycles at each temporal step and truncation error in the final temporal step $t = 0.04$

τ (#)	0.02(2)			0.005(8)			0.00125(32)		
N	Jacobi iterations	Multigrid cycles	Truncation error	Jacobi iterations	Multigrid cycles	Truncation error	Jacobi iterations	Multigrid cycles	Truncation error
128	**7**	**4**	$\mathbf{1.6 \cdot 10^{-2}}$	4	3	$1.5 \cdot 10^{-2}$	3	2	$1.5 \cdot 10^{-2}$
256	21	5	$4.9 \cdot 10^{-3}$	**7**	**4**	$\mathbf{4.4 \cdot 10^{-3}}$	4	3	$4.3 \cdot 10^{-3}$
512	191	6	$1.9 \cdot 10^{-3}$	22	5	$1.2 \cdot 10^{-3}$	**7**	**4**	$\mathbf{1.2 \cdot 10^{-3}}$
1024	4300	7	$1.18 \cdot 10^{-3}$	184	6	$3.4 \cdot 10^{-4}$	22	5	$2.8 \cdot 10^{-4}$

From Table 2, we conclude the following:

- Both methods, Jacobi and multigrid, have an optimal computational complexity because the number of Jacobi iterations and multigrid cycles remains constant as the system is scaled in a realistic way.
- The constant number of Jacobi iterations and multigrid cycles depends on the h^2/τ ratio as it is predicted in Eq. (19) and Eq. (21) respectively.
- For some spatial and temporal stepsizes, the Jacobi method is faster than the multigrid method (a multigrid cycle costs more than a Jacobi iteration). But, in general, the number of iterations needed by the Jacobi method for converging the solution to the level of truncation is higher than the number of multigrid cycles. The number of Jacobi iterations (19) grows with N faster than the number of multigrid cycles (21).

5. Scalability of multigrid methods

In this section we study the scalability of the d-dimensional FMVJ method on a d-dimensional processor mesh using the scalability analysis proposed in Section 3. In Section 5.1, we present an analytical study of the parallel system. By means of this study it is possible to analyze its execution time, efficiency and scalability. The analytical results are experimentally validated for the two-dimensional case in Section 5.2.

5.1. Analytical study

Performance prediction can provide a good understanding of the algorithm-architecture combination. Many factors determine the behaviour of a parallel system. This makes a detailed study complicated. However, a deterministic analytical prediction of the execution time and efficiency of the parallel system is useful for:

- Understanding the interactions between the algorithm and the architecture.
- Comparing two algorithms on the same architecture, or the same algorithm on different architectures.
- Analyzing the scalability of the parallel system. It is possible to obtain the isoefficiency and isotime functions with the execution time and the efficiency functions.
- Studying the effects of the architecture parameters on the execution time, efficiency and scalability.

We consider that a d-dimensional processor mesh is used to implement the d-dimensional FMVJ method. We assume that each processor has $2d$ bidirectional channels to connect with its neighbouring processors and these channels communicate in parallel. The time to send a message of n data to a neighbouring processor varies as $t_{\text{start}} + nt_{\text{com}}$, where t_{start} is the start time to send the message and t_{com} is the time per data. The t_{cpu} parameter denotes the time to perform a floating point arithmetic operation. These parameters depend on the architecture.

5.1.1. Parallel execution time of an MVJ-cycle

An MV-cycle is a special case of the MVJ-cycle, which descends until a lower level with two spatial steps per side ($N_{\text{lower}} = 3$). Let us suppose n_1 and n_2 are smoothing iterations in the ascending and descending parts of the multigrid cycle, a_1 denotes the cost of the restriction, prolongation and residual operators divided by the cost of a smoothing iteration, and $s'N^d t_{\text{cpu}}$ is the cost of a smoothing iteration on a N^d point grid. Then the sequential computational cost of an MV-cycle [3,15] is given by:

$$T^{\text{MV}}(N^d,1) = (n_1 + n_2 + a_1) s' \left(N^d + \frac{N^d}{2^d} + \dots \right) t_{\text{cpu}}$$
$$\approx (n_1 + n_2 + a_1) s' \frac{N^d}{(1 - 2^{-d})} t_{\text{cpu}} = \frac{c_1 s'}{(1 - 2^{-d})} N^d t_{\text{cpu}}. \tag{23}$$

The sequential computational cost in an MVJ-cycle is obtained subtracting to the last expression the cost given by the same expression in the lower level, and adding the result to the cost of γ_1 Jacobi iterations on the lower level. Since the computational cost of a Jacobi iteration is $sN^d t_{\text{cpu}}$, then the computational cost of an MVJ-cycle is given by the following expression:

$$T^{\text{MVJ}}(N^d,1) = T^{\text{MV}}(N^d,1) - T^{\text{MV}}\left(N^d_{\text{lower}},1\right) + \gamma_1 T^{\text{Jacobi}}\left(N^d_{\text{lower}},1\right)$$
$$= \left(\frac{c_1 s'}{(1 - 2^{-d})} \left(N^d - N^d_{\text{lower}} \right) + \gamma_1 s N^d_{\text{lower}} \right) t_{\text{cpu}}. \tag{24}$$

The degree of parallelism is always higher than the number of processors because the lower grid level is above the critical one (grid level with a number of spatial steps equal to the number of processors). Thus, the parallel computational time on a p^d processor mesh is the sequential time (24) divided by p^d.

Following this, we estimate the communication time. The processors have to communicate their boundary points to the $2d$ neighbouring processors after each smoothing iteration, restriction, prolongation and residual. Then, $2d$ communications of size $(N/p)^{d-1}$ are performed in parallel (we consider all channels working in parallel). If a_2 denotes the number of communications of the restriction, prolongation and residual operators, the communication time, supposing that the lower level is the critical one, is given by:

$$T_{\text{com}}\left(N^d,p^d\right) = (n_1 + n_2 + a_2) \left(t_{\text{start}} + \left(\frac{N}{p} \right)^{d-1} t_{\text{com}} + t_{\text{start}} + \frac{(N/p)^{d-1}}{2^{d-1}} t_{\text{com}} \right.$$
$$\left. + t_{\text{start}} + \frac{(N/p)^{d-1}}{2^{2(d-1)}} t_{\text{com}} + \dots \right)$$
$$\approx c_2 \log_2 N t_{\text{start}} + \frac{c_2}{\left(1 - 2^{-(d-1)}\right)} \frac{N^{d-1}}{p^{d-1}} t_{\text{com}}. \tag{25}$$

The communication time in an MVJ-cycle, with a lower level above the critical one, is obtained subtracting to the last expression the communication time given by the same expression in the lower level, and adding the result to the communication time of γ_1 Jacobi iterations:

$$T_{\text{com}}^{\text{MVJ}}(N^d,p^d) = T_{\text{com}}(N^d,p^d) - T_{\text{com}}(N_{\text{lower}}^d,p^d) + \gamma_1 T_{\text{com}}^{\text{Jacobi}}(N_{\text{lower}}^d,p^d)$$
$$= \left(c_2 \log_2 \frac{N}{N_{\text{lower}}} + \gamma_1\right) t_{\text{start}}$$
$$+ \left(\frac{c_2}{(1-2^{-(d-1)})} \frac{N^{d-1} - N_{\text{lower}}^{d-1}}{p^{d-1}} + \gamma_1 \frac{N_{\text{lower}}^{d-1}}{p^{d-1}}\right) t_{\text{com}}. \tag{26}$$

Adding Eq. (26) to Eq. (24) divided by p^d, the global execution time $T^{\text{MVJ}}(N^d,p^d)$ is obtained.

5.1.2. Parallel execution time of the FMVJ method

*

The computational cost of the FMVJ method is the cost of n_3 MVJ-cycles on each grid level, from the higher to the lower plus one, plus the cost of γ_2 Jacobi iterations on the lower level:

$$T^{\text{FMVJ}}(N^d,1) = n_3 \Sigma_{M=2N_{\text{lower}},\dots,N} T^{\text{MVJ}}(M^d,1) + \gamma_2 T^{\text{Jacobi}}(N_{\text{lower}}^d,1)$$
$$= \left(n_3 \frac{c_1 s'}{(1-2^{-d})}\left(\frac{N^d - N_{\text{lower}}^d}{(1-2^{-d})} - \log_2 \frac{N}{N_{\text{lower}}} N_{\text{lower}}^d\right)\right.$$
$$\left. + \left(\gamma_1 n_3 \log_2 \frac{N}{N_{\text{lower}}} + \gamma_2\right) s N_{\text{lower}}^d\right) t_{\text{cpu}}. \tag{27}$$

Since this method is perfectly parallel, the parallel computational time is the sequential time (27) divided by the number of processors, p^d.

The communication time is obtained in an analogous way:

$$T_{\text{com}}^{\text{FMVJ}}(N^d,p^d) = n_3 \Sigma_{M=2N_{\text{lower}},\dots,N} T_{\text{com}}^{\text{MVJ}}(M^d,p^d) + \gamma_2 T_{\text{com}}^{\text{Jacobi}}(N_{\text{lower}}^d,p^d)$$
$$= \left(n_3 c_2 \log_2^2 \frac{N}{N_{\text{lower}}} + n_3 \gamma_1 \log_2 \frac{N}{N_{\text{lower}}} + \gamma_2\right) t_{\text{start}}$$
$$+ \left(\frac{n_3 c_2}{(1-2^{-(d-1)})}\left(\frac{N^{d-1} - N_{\text{lower}}^{d-1}}{(1-2^{-(d-1)})p^{d-1}} - \frac{N_{\text{lower}}^{d-1}}{p^{d-1}} \log_2 \frac{N}{N_{\text{lower}}}\right)\right.$$
$$\left. + n_3 \gamma_1 \frac{N_{\text{lower}}^{d-1}}{p^{d-1}} \log_2 \frac{N}{N_{\text{lower}}} + \gamma_2 \frac{N_{\text{lower}}^{d-1}}{p^{d-1}}\right) t_{\text{com}}. \tag{28}$$

Adding Eq. (28) to Eq. (27) divided by p^d, the global execution time $T^{\text{FMVJ}}(N^d,p^d)$ is obtained.

5.1.3. Isoparameter functions of the FMVJ method on a mesh of processors

The isoefficiency function is the way the problem size N^d needs to grow with p^d to maintain a fixed efficiency. Efficiency is given by the following expression:

$$E^{\mathrm{FMVJ}}(N^d, p^d) = \frac{T^{\mathrm{FMVJ}}(N^d, 1)}{T^{\mathrm{FMVJ}}(N^d, 1) + T_{\mathrm{com}}^{\mathrm{FMVJ}}(N^d, p^d) p^d}. \tag{29}$$

The isoefficiency function is obtained equating $T^{\mathrm{FMVJ}}(N^d, 1)$, given in (27), with $T_{\mathrm{com}}^{\mathrm{FMVJ}}(N^d, p^d) p^d$, given in (28), and then solving this equation to find N^d as a function of p^d. It is difficult to find N^d as a function of p^d. However, if we consider that the number of points in the lower level, N_{lower}^d, grows linearly with the number of processors, p^d, the isoefficiency function grows linearly. Otherwise, if the lower level remains constant, the problem size N^d needs to grow faster to keep the efficiency unchanged.

In the same way, the execution time, Eq. (28) plus Eq. (27) divided by p^d, remains constant if the numbers of points in lower and higher levels are increased linearly with the number of processors. Therefore, isoefficiency and isotime functions are linear functions. This means that the parallel system is perfectly scalable, its scalability metrics are equal to one. Using a realistic scaling for obtaining more accurate solutions with all the memory space available the efficiency and the execution time remain constant.

The residual and initial systems are solved on the lower level using γ_1 and γ_2 Jacobi iterations respectively. We have considered these factors as constants because the system is scaled in a realistic way (Section 4.2). When the system is not scaled in a realistic way, γ_1 and γ_2 grow with N_{lower}, and therefore the system is not perfectly scalable.

5.2. Experimental results

The two-dimensional FMVJ method has been implemented on a transputer-based MIMD architecture (Parsys Supernode1000) using the message-passing programming model. Parallel C (ANSI C with explicit message-passing) was used to write the application. The Transputer-805 is a 2.2 Mflops microprocessor with four bidirectional links to connect it to other transputers, all in a single chip. Each interconnection link supports a rate of 10 Mbits/sec. The Supernode allows arbitrary networks of transputers to be configured, within the limitations of its four links. Larger machines can be configured with the Supernode as a unit of replication, in regular arrays or in high-level Supernodes. Our Transputer-805 network was configured as a two-dimensional mesh of processors.

This parallel application is being used to perform simulations of the NLS and MB equations. In this section we do not describe physical results but scalability and performance results. We study the execution time and efficiency when the parallel system is scaled using the time-critical and accuracy-critical scaling models. Physical results of the parallel application can be found in [12].

As it is shown in Fig. 9 for two problem sizes, if the parallel system is scaled using the time-critical scaling model, the problem size remains constant and, so, efficiency

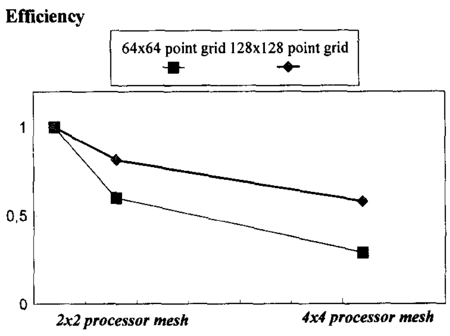


Fig. 9. Efficiency for two problem sizes.

decreases. This can be verified using Eq. (29): If N^d remains constant with an increasing number of processors, p_d, efficiency function decreases.

Fig. 10 shows the efficiency when the time-critical and isomemory scaling models are used to increase the parallel system. When the numbers of points in the higher and lower levels are increased linearly with the number of processors (isomemory scaling model), efficiency remains constant. The parallel execution times for both models are shown in Fig. 11. The execution time also remains constant with the isomemory model. Therefore, isoefficiency and isotime functions grow linearly. These experimental results verify the scalability analysis presented in Section 5.1.

The isomemory function maintains a constant efficiency and execution time because it corresponds to the isoefficiency and isotime functions. The FMVJ algorithm is therefore perfectly scalable on a processor mesh: If we increase the problem size with the number of processors, we obtain a more accurate solution with the same efficiency in the same execution time.

We conclude this section with an important consideration. A simulation consists of solving one system of equations per temporal step until the final temporal instant is

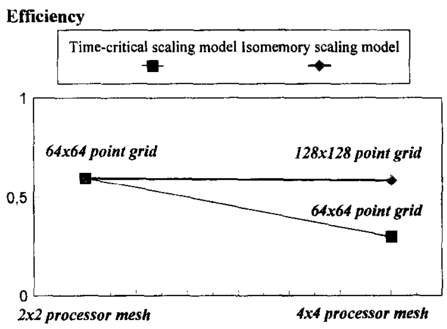


Fig. 10. Efficiency using time-critical and isomemory scaling models.

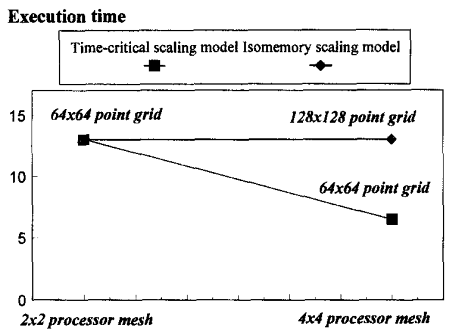


Fig. 11. Execution time using time-critical and isomemory scaling models.

reached. We have studied the scalability of the FMVJ method on a mesh of processors solving one of these systems

Let us study the scalability of the whole simulation. If the spatial stepsize is divided by 2 the temporal stepsize is divided by 4. Then, the number of temporal steps to reach the same temporal instant must be multiplied by 4, and, so, 4 times more systems of equations must be solved. This does not affect the isoefficiency function. However, N has to grow like $p^{d/d+2}$ to maintain a constant execution time. In this case, the higher the dimension is, the more scalable the parallel system is.

6. Conclusions

In this work we have described the use of multigrid methods to obtain numerical solutions to the nonlinear Schrödinger and Maxwell–Bloch equations on a mesh of processors. A full multigrid method is used to solve the systems of equations that arise at each temporal step when an implicit numerical scheme is employed to study these mathematical models from nonlinear optics.

A new scalability analysis of scientific applications on parallel architectures has been proposed. It considers execution time and efficiency as the parameters that the parallel system must preserve as it is scaled. In running scientific applications, we consider the following ways to scale the parallel system: time-critical and accuracy-critical scaling models. In the first case, scalability is assessed in terms of efficiency, or the execution time, as the number of processors increases. In the second case, the problem size grows with the number of processors, and one of the following parameters is held constant: execution time with the isotime function (isotime scaling model), efficiency with the isoefficiency function (isoefficiency scaling model), or average memory with the isomemory function (isomemory scaling model).

Using the accuracy-critical model, we are interested in obtaining the most accurate solution without exceeding the system's memory limits. That is, the problem size has to grow linearly with respect to the number of processors (isomemory function). In the best case, therefore, the isotime and isoefficiency functions grow linearly. This gives us a way to study the scalability of the parallel system qualitatively.

Isoefficiency and isotime scaling models have their own scalability metric. These measure how the scaling model makes the other parameter worse. The scalability of a parallel system is given by the two metrics. These metrics give us a way to study the scalability of a parallel system quantitatively. The system is perfectly scalable when both metrics are equal to one. This means that isoefficiency and isotime functions are linear functions: Employing a realistic scaling to obtain more accurate solutions with all the memory space available, the efficiency and the execution time remain unchanged.

In running scientific applications not only the problem size has to be scaled, other parameters must be scaled simultaneously to obtain more accurate solutions. The realistic scaling improves the algorithm complexity of some iterative methods, when they are applied for solving the system of equation that arises at each temporal step when some time dependent partial differential equations are approximated using an implicit numerical scheme. Moreover, due to this, Jacobi, multigrid and full multigrid methods are perfectly scalable on a mesh of processors. Their isoefficiency and isotime functions grow linearly with the number of processors. This is proved using an analytical prediction of its efficiency and execution time, and using experimental results of the parallel application.

Acknowledgements

This work has been supported by the Spanish research grants TAP94-0832-C02-01 and TIC94-0725-C03-02 and the Human Capital Mobility Network CHRX-CT-0459.

References

[1] A. Asenov, D. Reid and J.R. Barker, Speed-up of scalable iterative linear solvers implemented on an array of transputers, *Parallel Comput.* 20 (1994) 385–397.

[2] A. Brandt, Multi-level adaptive solutions to boundary-value problems, *Math. Comput.* 31 (1977) 343–400.

[3] W.L. Briggs, *A Multigrid Tutorial* (SIAM, Philadelphia, 1988).

[4] T.F. Chan and R. Schreiber, Parallel networks for multigrid algorithms: Architecture and complexity, *SIAM J. Sci. Stat. Comput.* 6/3 (1985) 698–711.

[5] T.F. Chan and Y. Saad, Multigrid algorithms on the hypercube multiprocessor, *IEEE Trans. Comput.* C-36/11 (1986) 969–977.

[6] Z. Fei, I Mártín, V.M. Pérez, F. Tirado and L. Vázquez, Numerical simulations and parallel implementation of some nonlinear Schrödinger systems, *Proceedings of Nonlinear Coherent Structure in Physics and Biology*, Bayreuth, Germany, June 1993 (Plenum Press, New York, 1995) 287–298.

[7] D. Gannon and J.V. Rosendale, Highly parallel multigrid solvers for elliptic PDEs: An experimental analysis, *Int. Rep.* 82-37, ICASE, NASA Langley Research Center, Hampton, VA, 1982.

[8] A. Gupta, V. Kumar and A. Sameh, Performance and scalability of preconditioned conjugate gradient methods on parallel computers, *IEEE Trans. Parallel and Distributed Systems* 6/5 (1995) 455–469.

[9] A. Hasegawa, *Optical Solitons in Fibers* (Springer Verlag, Berlin, 1990).

[10] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, and Programmability* (McGraw-Hill, New York, 1993).

[11] J. Linden, G. Lonsdale, H. Ritzdorf and A. Schüller, Scalability aspects of parallel multigrid, *Future Generation Computer Systems* 10 (1994) 429–440.

[12] I. Martín, V.M. Pérez, J.M. Guerra, F. Tirado and L. Vázquez, Numerical simulations of the Maxwell–Bloch laser equations, *Proceedings of Fluctuation Phenomena: Disorder and Nonlinearity*, El Escorial, Spain, September 1994 (World Scientific Publishing Co, Singapore, 1995) 400–406.

[13] I. Martín, F. Tirado, L. Vázquez and V.M. Pérez, Numerical aspects and solution of some nonlinear Schrödinger systems on a distributed memory parallel computer, *Proceedings of the 2nd Euromicro Workshop on Parallel and Distributed Processing*, Malaga, Spain, January 1994 (IEEE Computer Society Press, 1994) 245–252.

[14] O.A. McBryan, P.O. Frederickson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C. Thole and U. Trottenberg, Multigrid methods on parallel computers – A survey of recent developments, *Impact of Computing in Science and Engineering* 3 (1991) 1–75.

[15] S. F. McCormick, *Multigrid Methods* (SIAM, Philadelphia, 1987).

[16] J.M. Ortega, and R.G. Voigt, Solution of partial differential equations on vector and parallel computers, *SIAM Rev.* 27/2 (1985) 149–241.

[17] J.P. Singh, J.L. Hennesy and A. Gupta, Scaling parallel programs for multiprocessors: Methodology and examples, *IEEE Computer* 26/7 (1993) 42–50.

[18] X.H. Sun, The relation of scalability and execution time, *Int. Rep.* 95-62, ICASE, NASA Langley Research Center, Hampton, VA, 1995.

[19] X.H. Sun and D.T. Rover, Scalability of parallel algorithm-machine combinations, *IEEE Trans. Parallel and Distributed Systems* 5/6 (1994) 599–613.

[20] T.R. Taha and M.J. Ablowitz, Analytical and numerical aspects of certain nonlinear evolution equations, *Journal of Computational Physics* 55 (1984) 203–241.

[21] P. Wesseling, Introduction to multigrid methods. *Int. Rep.* 95-11, ICASE, NASA Langley Research Center, Hampton, VA, 1995.