Space-time Concurrent Multigrid Waveform Relaxation

Stefan G. Vandewalle
Eric F. Van de Velde

CRPC-TR93308
April 1993

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

CRPC-93-2 April 8, 1993

Space-time Concurrent Multigrid Waveform Relaxation*

Stefan G. Vandewalle[1] and Eric F. Van de Velde[2]

[1]*Department of Computer Science*
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium

[2]*CRPC and Applied Mathematics*
California Institute of Technology
Mail Code 217-50
Pasadena, California 91125

* *This work was supported in part by the NSF under Cooperative Agreement No. CCR-9120008. The government has certain rights in this material.*

Space-time Concurrent Multigrid Waveform Relaxation

Stefan G. Vandewalle[*] Eric F. Van de Velde[†]

Abstract. Multigrid waveform relaxation is an algorithm for solving parabolic partial differential equations on multicomputers. It is shown in this paper that the algorithm allows a partitioning of the computational domain into space-time blocks, i.e., subdomains of the space-time grid that are treated concurrently by different processors. The space-time concurrent multigrid waveform relaxation method is compared to two methods that use spatial concurrency only: space-concurrent multigrid waveform relaxation and standard time-stepping. It is illustrated that the use of space-time concurrency enables one to harness the computational power available on large-scale multicomputers. Timing results obtained on an Intel iPSC/2, an Intel iPSC/860 and the Intel Touchstone Delta are presented.

Keywords. parabolic partial differential equation, waveform relaxation, parallel computing, multigrid.

1 Introduction

Consider a system of ordinary differential equations obtained by semi-discretization of a parabolic initial boundary value problem,

$$\frac{d}{dt}u^h(t) = L^h u^h(t) + f^h(t)\ , \quad u^h(0) = u_0^h\ , \quad t \in [0,T]\ . \tag{1}$$

The value h is a measure of the mesh size, e.g., the largest distance between adjacent grid lines or grid points. $u^h(t)$ is the unknown function, whose components $u_i^h(t)$ approximate the solution of the partial differential equation at grid points x_i. The operator L^h is derived by spatial discretization of an elliptic operator, and $f^h(t)$ is a known function constructed from the boundary conditions or from a known source function. The computational complexity of solving (1) by a standard time-integration method is proportional to the number of time steps and grows linearly or superlinearly with the number of spatial discretization points.

[*]Senior Research Assistant of the Belgian National Fund for Scientific Research (N.F.W.O.), Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Leuven, Belgium.

[†]California Institute of Technology, Applied Mathematics 217-50, Pasadena, CA 91125.

The computational complexity rapidly becomes intractable when a high accuracy or an integration over a long time-interval is required. The computation of the solution to (1) is therefore an obvious candidate for implementation on a multicomputer.

Standard parallel approaches for solving systems of ordinary differential equations usually use *parallelism across the system*, also called *parallelism across space* when the equations are derived from a partial differential equation. In a time-stepping method spatial parallelism amounts to parallelizing the linear algebra on each time-level, e.g., the explicit update in an explicit scheme, or the direct or iterative linear system solver in an implicit scheme [15]. Time-stepping algorithms have the disadvantage of being fundamentally non-scalable. The parallelism in the algorithms does not scale linearly with the problem size, i.e., the total number of grid points in space and time. A doubling of the number of grid points, e.g., in order to increase the length of the integration interval, or in order to improve the accuracy of the approximation, usually doesn't allow one to double the number of processors. At any given moment in the computation all processors are active on the same time-level. Consequently, for a given spatial problem size, there is maximum number of processors that can be used effectively. This maximum is independent of the number of time steps that have to be computed. A truly scalable algorithm for solving ordinary differential equations will therefore have to combine parallelism across space with *parallelism across time*.

Practical experience with algorithms for ordinary differential equations that allow different processors to execute on different time-levels is rather limited. These algorithms are usually based on a Picard-like iteration or on the waveform relaxation method, see, e.g., [1, 2, 3, 4, 5]. In these methods time-parallelism is found in the quadrature methods and in the solvers of the linear recurrence relations underlying the ODE integrators. There is more experience with time-parallel algorithms for problems derived from time-dependent partial differential equations. The time-parallel PDE methods are often based on extensions of standard iterative methods for elliptic problems, like Jacobi, Gauss-Seidel and multigrid. These iterative methods are then adapted to operate concurrently on several time-levels. Some very impressive results have been reported in the literature, impressive both due to the complexity of the problems that can be solved (Navier-Stokes equations in [6, 7]), and due to the size of the multicomputers that can be used effectively (a 1024 processor Ncube in [16]). The time-parallel PDE methods, however, are less robust than their sequential counterparts. The convergence properties are strongly dependent on the discretization parameters, i.e., on time step and spatial grid size, and also on the number of time-levels treated concurrently. Their numerical efficiency in general deteriorates when the number of processors in the time-direction is increased. For a given problem size, i.e., a given space-time grid, convergence characteristics of the algorithms strongly depend on the way processors are allocated to the space- and time-dimensions.

In this paper we shall describe a scalable algorithm for solving equation (1), that uses parallelism both in space and in time. The algorithm has good convergence characteristics, that are independent of any particular allocation of processors. The algorithm extends the work on multigrid waveform relaxation described in [12, 13, 14]. The multigrid waveform relaxation method is briefly recalled in section 2. Two ways of introducing time-parallelism into this algorithm are presented in section 3. In section 4 we report experimental results obtained on a 64 processor Intel iPSC/2 multicomputer, a 128 processor Intel iPSC/860, and the 576 processor Intel Touchstone Delta. Finally, in section 5 we comment on the extension of our space-time concurrent method to nonlinear problems.

2 Multigrid Waveform Relaxation

A *multigrid waveform relaxation method* for solving (1) was presented in [8]. We briefly recall the two-grid cycle. It is a continuous-time iteration; it iterates with functions. Similar to classical two-grid methods it uses a *fine* grid, Ω^h, and a *coarse* grid, Ω^H. The former is the grid on which the solution of the problem is sought. The latter is used for calculating corrections to the approximations of the fine grid solution. Note that $\Omega^H \subset \Omega^h$, and that, typically, $H = 2h$. The two-grid cycle starts with an approximation, $\bar{u}^h$, to the solution on Ω^h. It determines a new approximation, $\bar{\bar{u}}^h$, in three steps: pre-smoothing, coarse grid correction, and post-smoothing. We shall only consider the case of an operator L^h that is linear. Let D^h be a diagonal matrix, and let A^h and B^h be strictly lower and upper triangular matrices, so that $L^h = A^h - D^h + B^h$. The two-grid method proceeds as follows.

• **Step 1: pre-smoothing.** Set $x^{(0)} = \bar{u}^h$. Perform ν_1 classical Gauss-Seidel waveform relaxation steps. For $\nu = 1, \dots, \nu_1$:

$$\frac{d}{dt}x^{(\nu)} + (D^h - A^h)\, x^{(\nu)} = B^h x^{(\nu-1)} + f^h \ , \quad x^{(\nu)}(0) = u_0^h \ , \quad t \in [0,T] \ . \tag{2}$$

Observe that the Gauss-Seidel nature of the splitting of L^h makes the above iteration computationally straightforward. The grid points are visited the one after the other. At each grid point the corresponding differential equation is solved as an equation in a single variable, using the most recently obtained approximations of the functions at the other grid points.

• **Step 2: coarse grid correction.** Compute the defect of approximation $x^{(\nu_1)}$:

$$d^h = \frac{d}{dt}x^{(\nu_1)} - L^h x^{(\nu_1)} - f^h = B^h(x^{(\nu_1-1)} - x^{(\nu_1)}) \ . \tag{3}$$

It can be verified that the error $x^{(\nu_1)} - u^h$, denoted by v^h, satisfies the following ordinary differential equation (the so-called *defect equation*):

$$\frac{d}{dt}v^h = L^h v^h + d^h \ , \quad v^h(0) = 0 \ , \quad t \in [0,T] \ . \tag{4}$$

Since (4) is of similar complexity as (1) we cannot expect to solve it in reasonable time. The two-grid cycle therefore continues by solving a coarse grid analogue to the defect equation.

$$\frac{d}{dt}v^H = L^H v^H + I_h^H d^h \ , \quad v^H(0) = 0 \ , \quad t \in [0,T] \ . \tag{5}$$

Its solution, v^H, defined on Ω^H, is an approximation to v^h. In (5), $I_h^H : \Omega^h \to \Omega^H$ is a *restriction* operator, and corresponds to calculating weighted averages of functions. L^H is the coarse grid equivalent to L^h. The coarse grid solution v^H is prolongated to Ω^h, using a waveform extension of a standard interpolation operator, $I_H^h : \Omega^H \to \Omega^h$, and approximation $x^{(\nu_1)}$ is corrected,

$$\bar{x}^h = x^{(\nu_1)} - I_H^h v^H \ . \tag{6}$$

• **Step 3: post-smoothing.** Perform ν_2 more smoothing relaxations of type (2), starting with $x^{(0)} = \bar{x}^h$, and set $\bar{u}^h = x^{(\nu_2)}$.

The two-grid cycle can be applied in recursive way to solve (5). This leads to a genuine *multigrid* waveform relaxation algorithm. The algorithm is usually combined with the idea of *nested iteration* or *full multigrid.* The problem is first solved solved on Ω^H. The coarse grid solution u^H is then interpolated to Ω^h and used as the starting iterate for one or more two-grid or multigrid cycles. The full multigrid interpolation step is as follows:

$$\bar{u}^h(t) := \bar{I}_H^h u^H(t) + (u_0^h - \bar{I}_H^h u_0^H) \ . \tag{7}$$

Here, $\bar{I}_H^h : \Omega^H \to \Omega^h$ is an interpolation operator, often of higher order than I_H^h. The second term in the interpolation formula is needed in order to satisfy the initial condition, $\bar{u}^h(0) = u_0^h$. This "correction" is not required in the full multigrid algorithm for solving elliptic problems.

For a convergence analysis of the method, we refer the reader to [8]. Extensive numerical experiments, a comparison to classical time-stepping schemes, and a description of how the algorithm can be extended to nonlinear, to time-periodic, and to systems of equations, is given in [12]. The use of multigrid waveform relaxation as a solver in fluid flow calculations, modelled by the Navier-Stokes equations, is discussed in [11].

3 Space-time Concurrency

3.1 Introduction

An implementation of the algorithm necessitates the choice of a representation for all variables, of a time-integration scheme and of a spatial discretization method. For simplicity's sake, we will, as in [12], only consider discretization with a constant and global time step determined a-priori. Spatial discretization is with finite differences on a regular mesh. The waveform method is, however, in no way limited to these choices. In fact, in [8] a numerical example is provided where the time step is chosen adaptively, during the computation, with different time steps from one grid point to the next.

Our choice leads us to solve a discrete problem defined on a regular space-time grid. Figure 1 illustrates three different grid partitioning techniques for a problem defined on a three-dimensional rectangular space-time grid (two space- and one time-dimension). The picture to the left depicts the standard partitioning as it is used in classical time-stepping schemes, [15]. The partitioning is in the spatial direction only, and is repeated every time step. The picture in the middle shows the partitioning that is used with the waveform relaxation method in [12, 13, 14]. Each processor is responsible for updating the functions

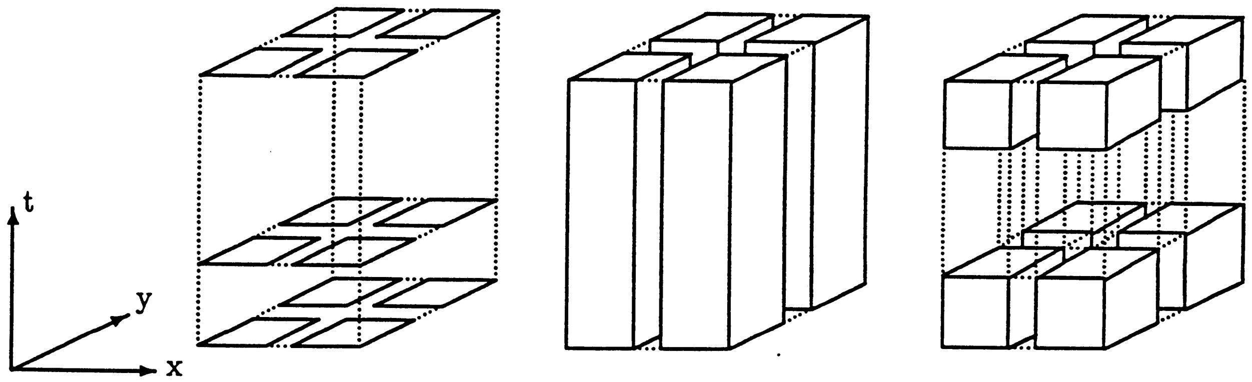


Figure 1: Grid partitioning used for parallelizing parabolic solvers on a two-dimensional rectangular domain: time-stepping method (left), space-concurrent waveform relaxation (middle), space-time concurrent waveform relaxation (right).

associated with a subset of the spatial domain. The time-direction is not partitioned. In the current paper we suggest the use of the partitioning graphically illustrated in the figure to the right. The partitioning divides the space-time grid in blocks that are treated concurrently by different processors.

The discrete-time analogues to the waveform relaxation operators operate on discrete functions, or vectors. It is straightforward to see that the restriction, prolongation, correction and defect calculation operators are entirely space- and time-concurrent. The main operation of the multigrid algorithm is the smoothing step. It is easily parallelized across space when a red/black ordering of grid points is used. Its parallelization across time is non-trivial because of the inherent sequential nature of the ODE-integration phase. However, the sequential part is limited to a small fraction of the computation.

Consider a two-dimensional spatial domain. At each grid point (x_i, y_j) an ordinary differential equation in a single unknown is to be solved,

$$\frac{d}{dt}u^h_{i,j}(t) = a^h_{i,j}(t)u^h_{i,j}(t) + w^h_{i,j}(t) \ , \quad u^h_{i,j}(0) = u_0(x_i, y_j) \ , \ t \in [0,T] \ . \tag{8}$$

The right-hand side function $w^h_{i,j}(t)$ is a linear combination of functions defined at (x_i, y_j) and neighbouring grid points. In the case of a five-point stencil, one obtains a formula like

$$\begin{aligned} w^h_{i,j}(t) &= a^h_{i,j-1}(t)u^h_{i,j-1}(t) + a^h_{i-1,j}(t)u^h_{i-1,j}(t) + a^h_{i+1,j}(t)u^h_{i+1,j}(t) \\ &\quad + a^h_{i,j+1}(t)u^h_{i,j+1}(t) + f^h_{i,j}(t) \ . \end{aligned} \tag{9}$$

Note that the calculation of $w^h_{i,j}$ is trivially time-concurrent. The use of a one-step time-discretization method for (8) leads to a first order recurrence relation extending in the positive time-direction. It is of the following form

$$u^h_{i,j,n} = b_{i,j,n}u^h_{i,j,n-1} + c_{i,j,n} \ , \quad n = 1, \ldots, N \ , \tag{10}$$

where "i, j, n" is an index in the space-time grid. It denotes a value at grid point (x_i, y_j, t_n). With the trapezoidal rule, for example, the coefficients $b_{i,j,n}$ and $c_{i,j,n}$ are determined by

$$b_{i,j,n} = \frac{1 + a^h_{i,j,n-1}\Delta t/2}{1 - a^h_{i,j,n}\Delta t/2} \quad \text{and} \quad c_{i,j,n} = \frac{(w^h_{i,j,n} + w^h_{i,j,n-1})\Delta t/2}{1 - a^h_{i,j,n}\Delta t/2} . \tag{11}$$

These coefficients can be calculated concurrently on all time-levels.

To conclude, the only operation in a time-concurrent multigrid waveform relaxation cycle that requires special attention is the computation of a linear recurrence.

3.2 The pipeline method

A first approach for calculating (10) extracts parallelism in a *pipeline* fashion. It is closely related to the *wavefront* approach mentioned in [4, 5], the *time-segment pipelining* method discussed in [10], and the *sequential smoothing* variant described in [6]. The computation at a grid point is started by a processor allocated to a block of grid points that borders time-level $t = 0$. This processor calculates the recurrence as far as possible, until it reaches the "upper" boundary of its space-time block. It then communicates the last computed value to processor that is adjacent in the positive time-direction. While the latter processor continues the calculation of the linear recurrence, the former starts a new recurrence computation at a different grid point, e.g., at a grid point of the same color. A third processor comes into play when the second processor reaches it space-time block boundary and communicates its last value to its upper neigbour. After having received the necessary value from processor one, the second processor continues with the second recurrence, while the first processor starts a third recurrence computation. When the time-dimension of the grid is partitioned into P blocks, it takes a total of $P-1$ steps before all processors are actively computing (different) recurrence relations.

The pipeline algorithm retains the nature of the sequential computation. The numerical results are independent of the number of processors. The algorithm is perfectly parallel, except for the "filling" and "emptying" of the pipeline. When the number of spatial grid points is small or when the number of processors assigned to the time-dimension is large, the pipeline effect may be substantial and may dominate the computation. There is no parallelism when only one equation is to be solved (e.g., on the coarsest grid in the multigrid method). The communication complexity of the algorithm is large. It requires a very large number (of the order of the number of spatial grid points) of small messages (one value).

3.3 The partition method

The partition method, also referred to as the algorithm of Wang, is a well-known method for the parallel solution of tridiagonal systems, see, e.g., [9, 12] and the references therein. A slight modification of the algorithm can be used for calculating (10) on a multicomputer. We shall first consider the case of a single recurrence.

Equation (10) can be rewritten as a system of linear equations with a bidiagonal coefficient matrix, with ones on the main diagonal and the $-b_{i,j,n}$ values on the subdiagonal. This matrix

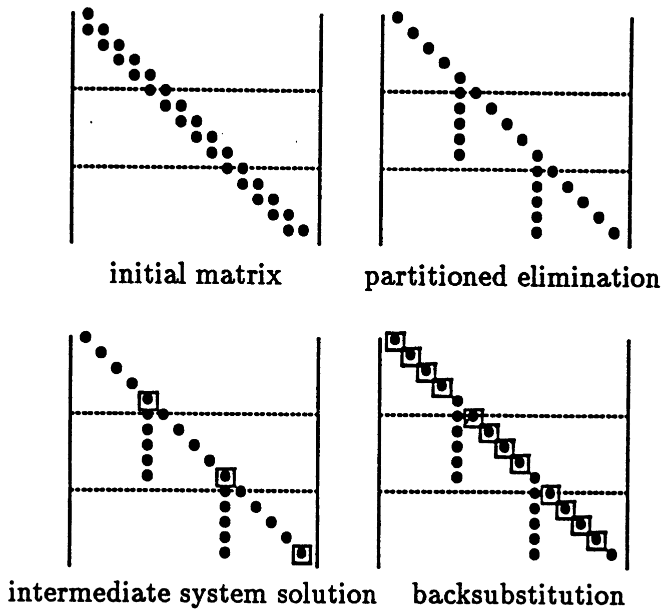


Figure 2: Three stages of the partition method applied to a bidiagonal matrix.

is graphically represented in the upper left part of Figure 2. Due to the grid partitioning in the time-dimension, the equations of the system are physically distributed onto different processors. Each processor contains one block of equations, in the figure separated by dotted lines. The algorithm proceeds in three steps. A first step, the partitioned elimination step, is entirely time-concurrent. In this step each processor locally eliminates its subdiagonal. This introduces some fill-in elements. It is easily checked that the resulting system is decoupled. An intermediate system with one equation per processor can be solved independently from the other equations. The unknowns that are calculated in this step are graphically indicated by squares, in the lower left figure. The intermediate system can be solved in various ways, all of which require interprocessor communication. The most straightforward approach is to calculate the unknowns sequentially, see [9]. This requires one receive and one send operation per processor. A different but more complex approach would be the use of a parallel cyclic reduction or recursive doubling method. The third step is a backsubstitution step, and is again completely time-concurrent. The complexity of the algorithm is analyzed in [9]. It can be checked that the algorithm requires $5N/P$ floating point operations per processor, whereas the sequential computation of (10) requires $2N$ operations. The achievable speedup, disregarding the effect of data transport and the computation in the intermediate system solution step, is therefore limited to $2/5P$.

The algorithm immediately extends to the case of multiple bidiagonal systems. To minimize communication overhead each step is applied to all systems before proceeding to the next step. The messages in the intermediate step are grouped together and sent as one large message. Note that the communication complexity is very different from the communication complexity in the pipeline method. The algorithm requires only a few large messages.

3.4 Remarks

The pipeline approach can be applied without change to time-integration methods different from simple one-step schemes. While this is in principle also possible for the partition method, in practice it suffers from an increased arithmetic complexity. It suffices to consider the use of a two-step method, which leads to a tridiagonal matrix with two subdiagonals.

Apart from the linear recurrences in the smoothing, there is one more operation that requires communication among processors assigned to different blocks of time-levels, namely the correction in the full multigrid interpolation (7). The correction term $u_0^h - \bar{I}_H^h u_0^H$ is calculated by the set of processors assigned to the space-time blocks that border the $t = 0$ time-level, and then broadcast in the time-dimension.

4 Experimental results

The numerical properties of the multigrid waveform relaxation method are discussed at great length in [12], where a large number of examples is given. These properties are not altered by the space-time concurrent implementation. In this paper we will therefore only consider the parallel characteristics of the method. We have implemented the space-time concurrent algorithm on three generations of Intel multicomputers: a 64 processor iPSC/2, a 128 processor iPSC/860, and the 576 processor Touchstone Delta. The knowledge of some of their hardware properties is required for a good understanding of the timing results that are given further below. The iPSC/2 and iPSC/860 are machines with hypercube topology, whereas the Delta is a two-dimensional mesh. The iPSC/2 nodes are Intel 386/387 processors with peak performance of 0.36 Mflops. The nodes of the iPSC/860 and the Delta consist of Intel i860 processors, with peak performance of 60 Mflops. The cost of sending a message of n bytes from one processor to another is modelled by the following formula, $T(n) = \alpha + \beta n + (l-1)\gamma$, where α is the startup-time, β the per byte transfer cost, l the number of links (*hops*) between the communicating processors, and γ the so-called hop-penalty. The values of these parameters are given in Table 1.

We consider solving the two-dimensional heat equation,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} , \quad (x, y) \in [0,1] \times [0,1] , \quad t \in [0,1] , \tag{12}$$

with Dirichlet boundary conditions that are derived from a known solution, which is given by $u(x, y, t) = 1 + \sin(\pi/2x)\sin(\pi/2y)e^{-\pi^2/4t}$. This problem is discretized in space with central differences on a regular rectangular grid with grid size h. Time-discretization is by the trapezoidal rule with a constant global time step Δt. Throughout the remainder of this section we will use the full multigrid algorithm with one multigrid V-cycle on each grid level. It is illustrated in [12] that this leads to a solution with an algebraic error smaller than the discretization error. That is to say, further iterations on the fine grid do not reduce the actual error of the approximation.

Tables 2 and 3 present execution times of the space-time concurrent full multigrid algorithm on different set of processors. (p_x, p_y, p_t) stands for the number of processors in x-, y-

and t-dimensions. The performance of the pipeline method is compared to the performance of the partition method. Notwithstanding its higher arithmetic complexity, the partition method proves to be the better choice. The pipeline method is competitive, only when the number of processors in the time-dimension is small. Table 3 also clearly illustrates the importance of choosing a partitioning of space *and* time.

In a second set of experiments we compare the execution time as a function of the number of processors for the three algorithms that were illustrated in Figure 2. The first algorithm is a parallel implementation of a standard time-stepping method based on the trapezoidal rule (Crank-Nicolson method). It applies one full multigrid cycle on each time-level. Details of the implementation are given in [12]. The second method is space-concurrent multigrid waveform relaxation. The third method is space-time concurrent multigrid waveform relaxation using the partition method and an experimentally determined "best" space-time grid partitioning. The three algorithms solve the same problem, on the same space-time grid, to a similar accuracy (algebraic error smaller than discretization error). Their implementation is of similar complexity, with a similar programming style and similar optimizations.

Figure 3 presents execution times on the iPSC/2 and iPSC/860 machines. Observe that the curve for the time-stepping method rapidly levels off because of the high communication complexity of the method. This is especially so on the iPSC/860, a machine with very unbalanced communication/computation hardware. The performance of the waveform methods is clearly superior. This is due to a dramatic reduction in communication requirements. In [14] it was shown that the waveform algorithm requires only a small fraction of the number of messages of the time-stepping method. The space-time concurrent method is more performant than the method with spatial concurrency only, because of better load-balancing properties. Load-imbalance is mainly caused by the impossibility of assigning an equal number of spatial grid points to each processor. The larger the number of processors in the spatial dimension, the more important this problem becomes. Note that even on a single processor waveform relaxation would outdo the time-stepping method, provided enough memory is available. This is due to a slightly smaller arithmetic complexity, see [12], due to smaller programming overheads (e.g., indexing and loop overheads), and (for the Intel i860 processor) due to the fact that operations on vectors usually run at a larger fraction of the peak Mflop rate than operations on scalars.

Similar curves for the Touchstone Delta are presented in Figure 4. It is especially interesting to compare the performance of the methods for the large problem ($h = 1/256$ and $\Delta t = 1/512$). The speedup obtained by the time-stepping method is about 19, whereas the (scaled) speedup obtained by the time-parallel method is about 320.

5 Concluding Remarks

The space-time concurrent method can be extended straightforwardly to nonlinear problems, by using the *full approximation scheme* described in [12]. The smoothing part of this algorithm consists of a Gauss-Seidel waveform relaxation *Newton* method. Each nonlinear differential equation is linearized once in every smoothing step, around the current approxi-

Table 1: Communication parameters: α: startup cost, β: byte transfer cost, γ: hop penalty (in μsec). The values between brackets are for small messages ($n \leq 100$ bytes).

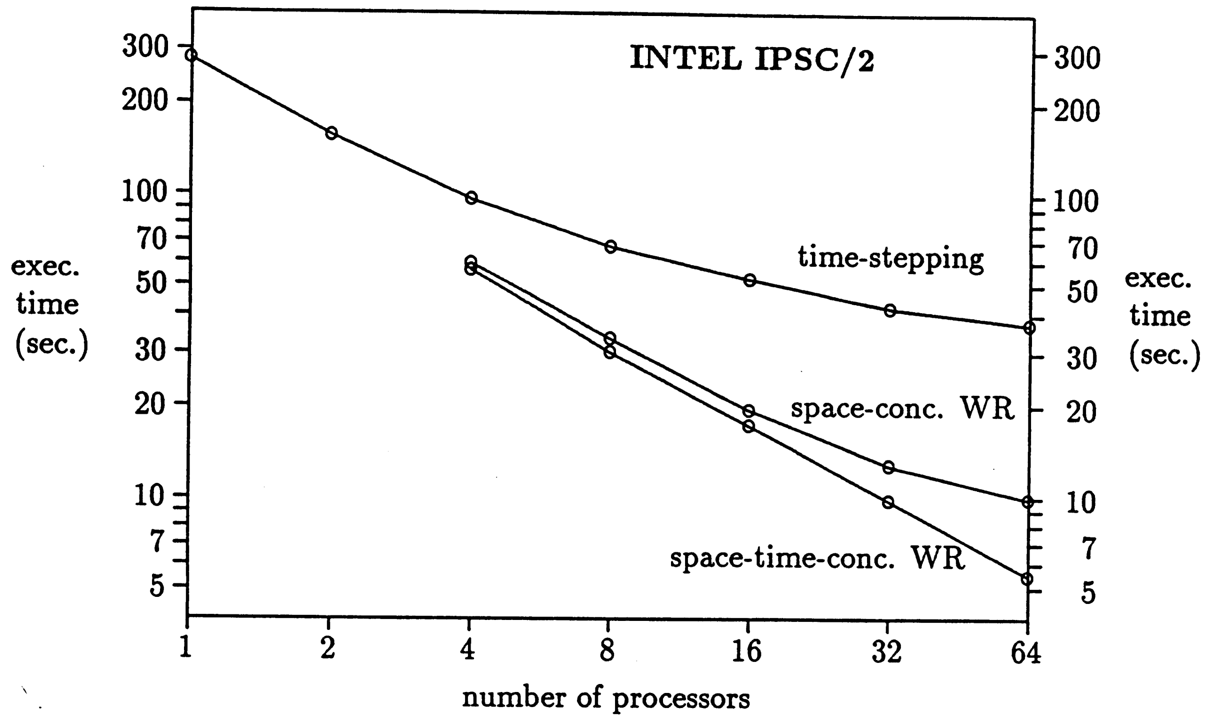
	α	β	γ
iPSC/2	700 (372)	0.4 (0.2)	33 (11)
iPSC/860	136 (75)	0.4 (0.2)	33 (11)
Delta	72	0.08	0.05

Table 2: Execution time, in seconds, for executing the full multigrid algorithm with different numbers of processors ($h = 1/64$, $\Delta t = 1/128$).

processors	Intel iPSC/2		Touchstone Delta	
$p_x \times p_y \times p_t$	partitioning	pipelining	partitioning	pipelining
$2 \times 2 \times 1$	58.5	58.5	3.80	3.80
$2 \times 2 \times 2$	33.7	29.9	2.26	2.83
$2 \times 2 \times 4$	17.7	18.3	1.22	1.86
$2 \times 2 \times 8$	9.76	11.6	0.73	1.79
$2 \times 2 \times 16$	5.77	8.40	0.50	1.71

Table 3: Execution time, in seconds, for executing the full multigrid algorithm with different processor configurations on a 64 processor machine ($h = 1/64$, $\Delta t = 1/128$).

processors	Intel iPSC/2		Touchstone Delta	
$p_x \times p_y \times p_t$	partitioning	pipelining	partitioning	pipelining
$8 \times 8 \times 1$	6.80	6.80	0.89	0.89
$4 \times 8 \times 2$	6.10	6.02	0.73	0.88
$4 \times 4 \times 4$	5.73	5.76	0.54	0.93
$2 \times 4 \times 8$	5.54	6.44	0.48	1.30
$2 \times 2 \times 16$	5.77	8.40	0.50	1.71
$1 \times 2 \times 32$	6.80	14.1	0.75	2.80
$1 \times 1 \times 64$	9.88	23.1	1.09	4.97



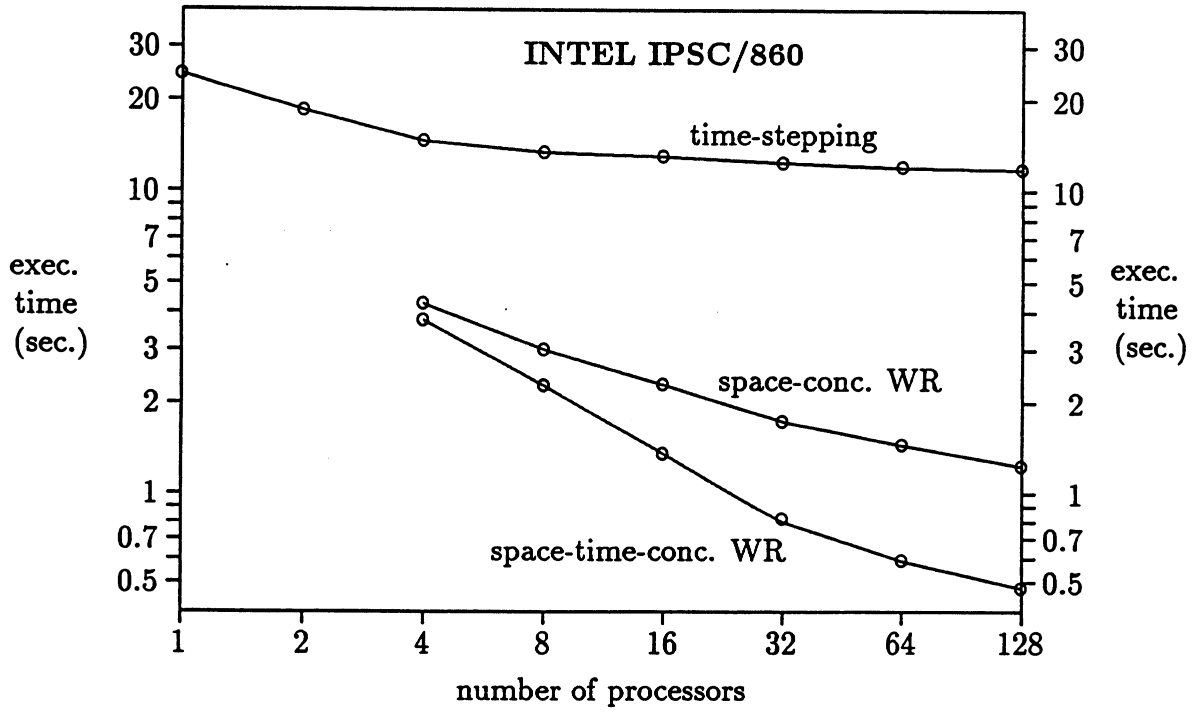
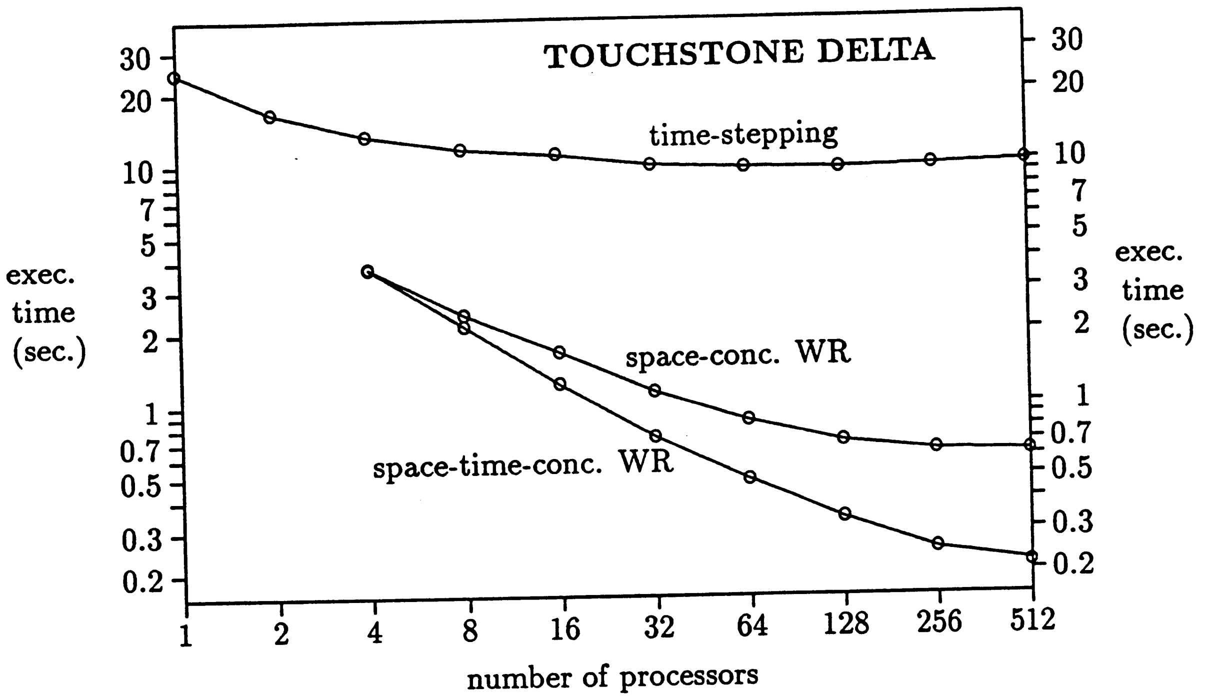


Figure 3: Execution time, in seconds, as a function of the number of processors, for executing the full multigrid algorithm ($h = 1/64$ and $\Delta t = 1/128$).



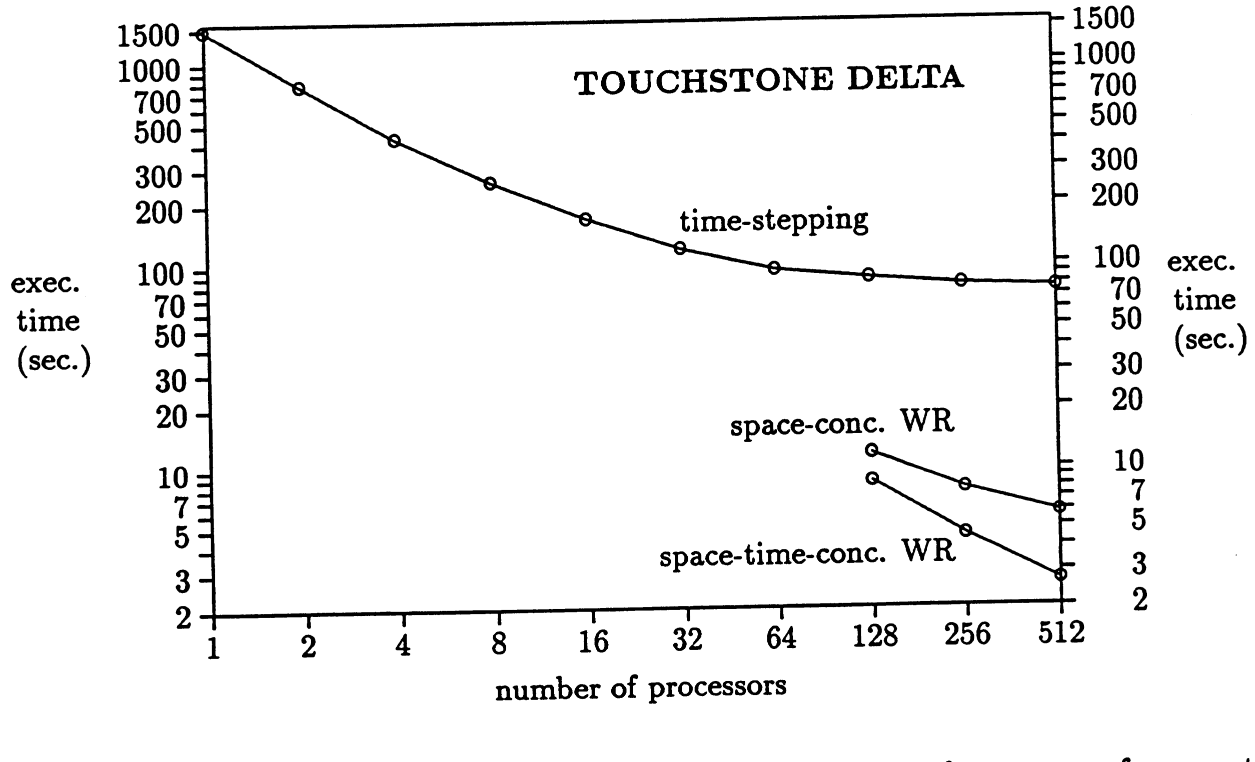


Figure 4: Execution time, in seconds, as a function of the number of processors, for executing the full multigrid algorithm (upper graph: $h = 1/64$ and $\Delta t = 1/128$, lower graph: $h = 1/256$ and $\Delta t = 1/512$, Touchstone Delta).

mation, and solved by a linear ODE-integrator. The computational complexity is therefore very similar to that of the linear solver described in this paper. The algorithm can be extended in a similar way to systems of (nonlinear) parabolic partial differential equations.

Note that the memory requirements of the waveform method are very different from those of the time-stepping scheme. In the latter it suffices to store only one or two time-levels, while the former requires storing the whole space-time grid. If not enough memory is available, the waveform method can still be used by subdividing time-interval $[0,T]$ into a sequence of *windows*, $[0,T_1],[T_1,T_2],\ldots,[T_k,T]$ that are treated sequentially. For optimal efficiency these windows should be chosen as large as possible. If memory is very limited, so that only one or two grid levels can be stored, the multigrid waveform relaxation method becomes equivalent to a standard time-stepping scheme.

Acknowledgments. The authors would like to thank the Mathematical Sciences Section, Division of Engineering Physics and Mathematics, Oak Ridge National Laboratory, for providing access to their Intel iPSC/2 and iPSC/860 machines, and the Caltech Supercomputing Consortium, for providing computing time on the Touchstone Delta.

References

[1] A. Bellen and F. Tagliaferro. A combined WR-parallel steps method for ODEs. In P. Messina and A. Murli, editors, *Parallel Computing: Problems, Methods and Application*, pages 77–86, New York, 1992. Elsevier Science Publishers B.V.

[2] A. Bellen and M. Zennaro. Parallel algorithms for initial value problems for difference and differential equations. *J. Comp. Appl. Math.*, 25:341–353, 1984.

[3] K. Burrage. Parallel methods for initial value problems. *Applied Numerical Mathematics*, 11:5–25, 1993.

[4] C.W. Gear. Waveform methods for space and time parallelism. *J. Comp. Appl. Math.*, 38:137–147, 1991.

[5] C.W. Gear and X. Xuhai. Parallelism across time in ODEs. *Applied Numerical Mathematics*, 11:45–68, 1993.

[6] G. Horton. *Ein zeitparalleles Lösungsverfahren für die Navier-Stokes-Gleichungen.* PhD-thesis, Universität Erlangen-Nünberg, 1991.

[7] G. Horton. *TIPSI - A Time-Parallel SIMPLE-Based Method for the Incompressible Navier-Stokes-Equations.* In K. Reinsch *et al.*, editors, *Parallel Computational Fluid Dynamics '91*, pages 243–256, New York, 1992. Elsevier Science Publishers B.V.

[8] Ch. Lubich and A. Ostermann. Multigrid dynamic iteration for parabolic equations. *BIT*, 27:216–234, 1987.

[9] P. Michielse and H. Van der Vorst. Data transport in Wang's partition method. *Parallel Computing*, 7:87–95, 1988.

[10] P. Odent. *Electrical-level simulation of VLSI MOS circuits using multi-processor systems.* PhD-thesis, Katholieke Universiteit Leuven, Belgium, January 1990.

[11] C.W. Oosterlee and P. Wesseling. Multigrid schemes for time-dependent incompressible Navier-Stokes equations. Report no. 92-102, Delft University of Technology, Faculty of Technical Mathematics and Informatics, 1992.

[12] S. Vandewalle. *Parallel Multigrid Waveform Relaxation for Parabolic Problems.* B.G. Teubner Verlag, Stuttgart, 1993.

[13] S. Vandewalle and R. Piessens. Numerical experiments with nonlinear multigrid waveform relaxation on a parallel processor. *Applied Numerical Mathematics*, 8(2):149–161, 1991.

[14] S. Vandewalle and R. Piessens. Efficient parallel algorithms for solving initial-boundary value and time-periodic parabolic partial differential equations. *SIAM J. Sci. Stat. Comput.*, 13(6):1330–1346, 1992.

[15] S. Vandewalle, R. Van Driessche, and R. Piessens. The parallel performance of standard parabolic marching schemes. *Int. J. High Speed Computing*, 3(1):1–29, 1991.

[16] D. Womble. A time-stepping algorithm for parallel computers. *SIAM J. Sci. Stat. Comput.*, 11(5):824–837, 1990.