Parallel Implementation of Non-Linear Evolution Problems using Parabolic Domain Decomposition *

appeared in Parallel Computing, Vol. 21, No. 7, pp. 1151-1183, 1995

A. Averbuch
School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel
M. Israeli   L. Vozovoi
Faculty of Computer Science, Technion, Haifa 32000, Israel

Abstract

We present implementation of parallel algorithms for the numerical solution of nonlinear time-dependent partial differential equations of parabolic type arising from complex large scale problems. The parallelization is achieved by using domain decomposition (DD) techniques. The essential feature of this algorithm is that the spatial discretization in each subdomain is performed by using spectral method with the Local Fourier Basis (LFB)[1]. Our solutions are based on a special projection technique that employed to localize functions in a smooth way on the extended subdomain.

The current paper continue the flow of our previous results [1, 4, 12, 13] on spectral multidomain algorithm. The application of the Parabolic Domain Decomposition (PDD) approach along with the LFB is shown to be very efficient when applied to a 2-dimensional domain splitted into strips and rectangular cells. In this case, *all* matching relations become completely uncoupled (at the price of some overlapping of subdomains required by the LFB implementation). Thus, all communication is reduced to interactions between neighbouring elements and thus fits to scalable message-passing multiprocessor.

The continuity of a global solution is attained by using a direct point-wise matching of the local subsolutions on the interfaces. The implementation of the LFB technique enables us to trade a 2-D problem with the overall coupling of the interface unknown into a set of uncoupled 1-D differential equations with simple matching relations.

2-D Navier-Stokes type modeling equation is implemented on the Meiko message-passing type scalable MIMD multiprocessor.

Detailed performance analysis is presented. The algorithm is scalable in the sense that problems of equal size have the same speedup when the number of processors increase because the communication is reduced from global to local and, thus, the solution depends on direct neighboring processors. When the size if each sub-domain in each processor is large enough a linear speedup is achieved. The implementation strategy of the algorithms can easily be changed to reflect the potential of having different resolution in each subdomain, which makes it valuable as an adaptive algorithm.

The same results are derived when the the alternate direction implicit (ADI) method is used as an efficient time-discretization scheme.

*This research was supported partly by a grant from the French-Israeli Binational Foundation for 1991-1992. The work of the second author was supported by the funds for the promotion of research at the Technion.
The work of the third author was also supported by a grant from Israel Ministry of Science and Technology for 1992-1993.

Key-words: Domain decomposition, parabolic PDEs, spectral method, local Fourier bases, Green function, MIMD multiprocessor, parallelization.
AMS(MOS) subject classification: 35K25, 35P05, 35R05

1 Introduction

We present parallel algorithms for the numerical solution of nonlinear time-dependent partial differential equations (PDE's) arising from complex large scale (intensive computing) problems in science and engineering, in particular, in the field of computational fluid dynamics. The solution is based on domain decomposition techniques. The essential feature of this algorithm is that the spatial discretization in each subdomain is performed by using spectral method with the Local Fourier Basis (LFB).

These parallel algorithms are based on our novel numerical approaches developed in series of works [1, 4, 12, 13] and the results in this paper demonstrate that the theoretical results are valid when these algorithms are implemented on a scalable message-passing multiprocessor.

High-performance computing technology has emerged as a powerful and indispensable aid to scientific and engineering research, product and process development, and all aspects of manufacturing. Such computational power can be achieved only by massively parallel computers. In the 1990's, parallel supercomputers are emerging in which a large number of CPUs are coupled together in a network to provide enormous computer power. High performance computing, sometimes called supercomputing, involves large scale computations to perform specific tasks. Significant increase in performance can sometimes be achieved by reducing large calculations to a number of smaller segments that can be executed in parallel.

Realization of these goals will lead to major gains in diversity of important applications, including aircraft/space vehicle design, computational fluid dynamics, pharmaceuticals/drug design, medical imaging, heart and lung research, robotics, signal/image processing, biotechnology, improved automation technology, economic modeling, seismic exploration, oil reservoir exploration, symbolic processing, databases, numerical weather prediction, numerical intensive computation and many more.

Multiprocessors become more and more indispensable facility for large scale scientific computing, in particular, for computational fluid dynamics [9]. For many years investigations have been carried out to understand the physical processes that occur in turbulent flows. Progress in the basic theory of turbulence translates nearly immediately into a wide range of engineering applications and technological advances (e.g. in aircraft industries, chemical engineering, weather forecasting, etc) that effect many aspects of everyday life. Laboratory visualizations of turbulent flows are particularly complicated and expensive. Therefore, the direct numerical simulations of fluid flows at high Reynolds numbers is a principle means of understanding and modeling turbulence.

Parallel and distributed systems under development offer the promise of inexpensive and massive computing power but only if appropriate algorithms can be implemented to avoid the bottlenecks of communication and synchronization as the number of processors increases.

Our solutions are based on a special projection technique, introduced in [2], that employed to localize functions in a smooth way on a range of an extended subdomain. The projection results in a periodic continuous function, as smooth as the original one, after which a spectral Fourier method is applied on local period to compute the elemental solutions concurrently in all subdomains.

The continuity of the global solution is attained by using direct point-wise matching of the local subsolutions on the interfaces. The exact matching relations are derived for the matching parameters in terms of the residuals of the subsolution and its first derivative on the interfaces.

The implementation of the LFB technique enables us to trade a 2-D problem with the overall coupling of the interface unknown into a set of uncoupled 1-D differential equations with simple matching relations. Thus, the numerical algorithm reduces to repeated implementation of 1-D routine, which is computational equivalent to the usual spectral Fourier algorithm, except for some extra operations that are needed to smooth the function.

The algorithm was tested on 2-D Navier-Stokes type modeling equation investigated in [6]. For time discretization we use a third order stiffly stable scheme with the implicit time stepping on the viscous time step.

Two types of domain decomposition are considered in the forms of parallel strips and rectangular cells where each uses a different connection schemes.

These algorithms are implemented on a massively parallel MIMD machine (message-passing type).

The implementation in this paper is based on periodic boundary conditions. In our next paper[3] we describe the implementation of the complete Navier-Stokes equation which is based on Dirichlet boundary condition.

To summarize, our parallel algorithm for the solution of CFD problems in multidomain regions presents a fast and direct method with high resolution in time and space. This algorithm requires the simplest parallel architecture where the communication is performed only between neighbouring subdomains.

The paper has the following structure: the emergence of parallel machines renewed interest in DD methods since they seem to give rise to efficient tools for parallelizing numerical algorithms and this is discussed in section 2. In section 3 we describe the theoretical background for time-dependent partial differential equations of parabolic type for which the influence of remote regions is practically negligible. We describe the model problem (section 3.1), the discretization in time (section 3.2), the discretization in space (section 3.3). Our main domain decomposition method is based on Parabolic Domain Decomposition (PDD) which is described in section 4. The matching between neighboring domain is given in section 4.1 and its extension to two dimensions is described in section 4.2. Detailed description of the strip and the cell computational algorithms is given in sections 5 and 6 respectively. These algorithms were implemented on the Meiko which is MIMD message-passing multiprocessor. This implementation is detailed in section 7 and its performance analysis is given in section 8. We also tried the alternate direction implicit (ADI) method as another efficient time-discretization scheme and its implementation and performance analysis is given in section 9.

2 Parallel Computers and Domain Decomposition Techniques

In the past several years computational algorithms based on domain decomposition (DD) techniques have attracted much attention because of their abilities to facilitate the solution of very large problems on multiprocessors in particular those that are classified as distributed scalable multiprocessors. They are also often used to design adaptive algorithms to capture steep gradients that appear in the solution of certain problems (e.g. inviscid Euler equations) or to enable the coupling of regions with different physical parameters (e.g. viscid and inviscid flows) in such way that a simplification of the mathematical problem is achieved. Another reason to resort to DD methods is to simplify problems in complicated geometries, when the mapping of the full domain into a simple shape (such as rectangle or disk) is difficult or it introduces artificial singularities or boundary layers into the solution in the transformed space. The emergence of parallel machines renewed interest in DD methods since they seem to give rise to efficient tools for parallelizing numerical algorithms.

After the decomposition the elemental problems can be decoupled and solved in each subdomain independently (to a great extent) except for a matching step, which is necessary for obtaining a smooth global solution from the different elemental solutions. The matching procedure requires communication among subdomains. This is a crucial step for DD algorithms because it introduces overheads in the implementation on a parallel computer and may, in severe cases, reduce or eliminate the gains of parallelization.

Massively parallel architectures, especially the flexible MIMD type message passing systems, are considered now as the most promising strategy in supercomputer hardware design. The parallel computers of MIMD type are scalable, i.e. in principle they are able to grow in accordance with the requirements of the task. Our proposed algorithms are scalable in the sense that problems of equal size consume the same processing time in each processor even when the number of processors increases because the communication is reduced from global to local and thus the solution depends on direct neighboring processors. Thus, we avoid spending time on communication which is the major bottleneck in speeding up numerical intensive computation on massively parallel multiprocessors.

The high performance of parallel machines can be realized and utilized to its upmost efficiency only if effective parallelizing algorithms are supplied. The geometric domain decomposition is intrinsically suited for the purposes of parallelization since the neighbouring pieces of space (subdomains) can be allocated to processors with short communication links. The original problem is solved independently in each processor, then a matching procedure is employed to enforce the continuity conditions on the interfaces. The objective of any DD method is to reduce the amount of interprocessor communications and data exchange to the minimum level possible to avoid communication and synchronization bottlenecks. This is essential to the type of problems addressed by this paper where domain decomposition, elemental spectral solution and globality of data exchange are linked together and thus, at first sight, they seem to be not optimal candidates to be ported to massively parallel machine.

3 Theoretical Background

Time-dependent partial differential equations of parabolic type are described for which the influence of remote regions is practically negligible. The locality property is exploited in domain decomposition techniques to simplify the global matching relations among subdomains.

The numerical solution of Partial Differential Equations has the potential for efficient parallelization on distributed message-passing type machines because partial derivatives represent local behaviour.

In this paper we concentrate mainly on local approaches for solving time-dependent problems of the parabolic type in multidomain regions. Detailed survey of some of principal domain decomposition approaches for solving elliptic and parabolic problems can be found in [5].

3.1 Model Problem

The algorithm was tested and implemented using the following model problem which is a two-dimensional equation of the Navier-Stokes type [10]

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \nabla^2 \mathbf{v} - \mathbf{N}(\mathbf{v}) + \mathbf{F}(\mathbf{x}, \mathbf{t}), \qquad \text{in } \Omega = (0, 2\pi)^2 \tag{3.1}$$

$$\nabla \cdot \mathbf{v} = 0$$

$$\mathbf{N}(\mathbf{v}) = (\mathbf{v}\cdot\nabla)\mathbf{v} + \frac{1}{2}(\nabla\cdot\mathbf{v})\mathbf{v} \tag{3.2}$$

$$\mathbf{v} = (v_x, v_y), \qquad \mathbf{F} = (F_x, F_y), \qquad \mathbf{x} = (x, y)$$

$$\mathbf{v}(x + 2\pi, y) = \mathbf{v}(x, y + 2\pi) = \mathbf{v}(x, y) \tag{3.3}$$

The non-linear term is written in the form (3.2) which preserves the energy.

3.2 Discretization (splitting) in time

The time discretization is performed by using a stiffly stable scheme investigated in [8]. It writes:

$$\frac{\gamma_0 \mathbf{v}^{n+1} - \sum_{q=0}^{J_i-1} \alpha_q \mathbf{v}^{n-q}}{\Delta t} = \nu\nabla^2\mathbf{v}^{n+1} - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.4}$$

where the coefficients γ_0 and α_q are the standard coefficients of the implicit stiffly stable schemes corresponding to the order J_i. The coefficients β_q for the explicit contributions are computed by the method of undetermined coefficients and employing Taylor series expansions. We used a 3-d order scheme $J_i = J_e = 3$ with the coefficients: $\gamma_0 = 11/6, \alpha_0 = 3, \alpha_1 = -3/2, \alpha_2 = 1/3, \beta_0 = 3, \beta_1 = -3, \beta_2 = 1$.

Thus, the time discretization procedure results in the following equation for the function $\mathbf{v}^{n+1}$ on the new time step:

$$\nabla^2\mathbf{v}^{n+1} - \lambda^2\mathbf{v}^{n+1} = \mathbf{f} \tag{3.5}$$

where $\lambda^2 = \gamma_0/(\nu\Delta t)$ and the non-homogeneous term in the RHS contains the terms evaluated in the previous time levels n, $n-1$ and $n-2$:

$$\mathbf{f} = -\frac{1}{\nu\Delta t}\sum_{q=0}^{J_i-1} \alpha_q \mathbf{v}^{n-q} + \frac{1}{\nu}\sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.6}$$

3.3 Spectral Multidomain Technique with the LFB- Discretization in Space

Space discretization (splitting) is done by using the pseudospectral method which is based on the Local Fourier Basis (LFB)[1]. We illustrate the basic technique on the one-dimensional Helmholtz equation. We compute the equation (3.5) where f is a known function. The interval L is divided into S equal subdomains of length $l = L/S$. The algorithm consists of two main steps. First, we construct the particular solutions in each subdomain independently by applying arbitrary boundary conditions at the interfaces. For a space discretization we use the LFB. The LFB is particularly suitable for spatial discretization in DD methods because nonperiodic functions are represented in this basis by rapidly converging series. Matrices of differential operators in the transformed space are diagonal (in contrast with full matrices for the Chebyshev or Legendre bases). Thus, in contrast to the ordinary Fourier basis, the LFB preserves the spectral accuracy when applied to (nonperiodic) elemental functions in subdomains. At the same time this basis possesses all the advantages of the usual trigonometric bases such as a uniform mesh of collocation points, diagonal matrices for the differential operators in the transform space and fast transform algorithm. After this step, the overall solution appears as a piecewise function with jumps on the interfaces. In the second step, we match all the elemental solutions by using the boundary Green's functions (solutions of the corresponding homogeneous problem).

We consider the solution of Eq. (3.5) in one of the subdomains $(a,b) \subset (0,L)$. The algorithm consists of two steps: (1) smoothing procedure across the boundaries (interfaces) to localize the functions in a smooth manner by transforming the function to a periodic form and (2) the application of the spectral Fourier method with the locally supported trigonometric basis.

To implement smoothing, we introduce the bell function $B(x)$, supported on the extended interval $a_1 < a < b < b_1$:

$$\begin{array}{ll} B^2(x) + B^2(2\bar{a} - x) = 1 & x \in [a_1, a] \\ B(x) = 1 & x \in [a, b] \\ B^2(x) + B^2(2\bar{b} - x) = 1 & x \in [b, b_1] \\ B(x) = 0 & x < a_1,\ x > b_1 \end{array} \tag{3.7}$$

where $\bar{a} = (a + a_1)/2,\ \bar{b} = (b + b_1)/2$. This function is equal to $B = 1$ inside the subdomain and smoothly decays out words over the distance $2\epsilon = b_1 - b = a - a_1$. Some particular forms of $B(x)$ were tested in [1].

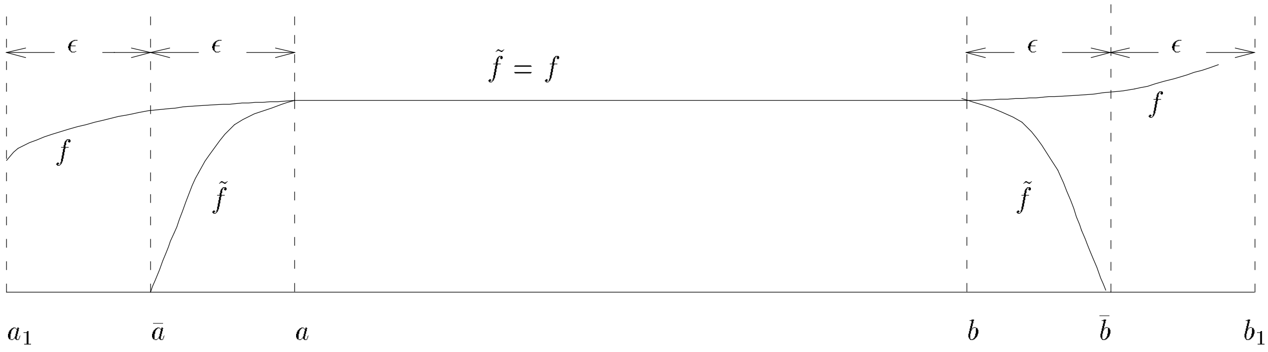

Figure 1: The folding operation

The bell function $B(x)$ is defined as:

$$B(x) = \begin{cases} 0, & x < a_1, x > b_1 \\ s(x), & x \in [a_1, a) \\ 1, & x \in [a, b] \\ c(x), & x \in (b, b_1] \end{cases} \tag{3.8}$$

where

$$s(x) = sin\theta(x), c(x) = \cos\theta(x);\ s^2(x) + c^2(x) = 1, \quad c(x) = s(-x)$$

$$\theta(x) = \int_{-\infty}^{x} \psi(\xi)d\xi, \quad \theta(x) + \theta(-x) = \frac{\pi}{2};$$

$$\int_{-\infty}^{+\infty} \psi(x)dx = \frac{\pi}{2}, \quad \sup\psi \subset (-\epsilon, \epsilon), \quad \psi(x) = \psi(-x)$$

The smoothing of the function f, denoted by $\tilde{f}$, appears as a "folding" across the lines $\bar{a}$ and $\bar{b}$ (see Figure 1):

$$\tilde{f} \equiv B \cdot f = \mathcal{F}_{\bar{a}}\mathcal{F}_{\bar{b}}f(x) = B(x)f(x) - B(2\bar{a} - x)f(2\bar{a} - x) - B(2\bar{b} - x)f(2\bar{b} - x) \tag{3.9}$$

(the "folded" function $\tilde{f}$ is defined in $[\bar{a}, \bar{b}]$; the second term is "switched on" only in the interval $x \in [a_1, a]$ and the third term in the interval $x \in [b, b_1]$, respectively). The extra pieces of the function f, required for the smoothing operation, are provided by overlapping of neighbouring subdomains over 4ϵ range. On the interval $x \in [a, b]$ we have $\tilde{f} = f$.

The smoothing procedure keeps the function $\tilde{f}$ highly continuous at $x = a, b$. In addition, Eq. (3.9) yields that in the vicinity of the points $x = \bar{a},\ x = \bar{b}$ the function $\tilde{f}(x)$ is odd and thus all even derivatives $\tilde{f}^{(2r)}(\bar{a}) = \tilde{f}^{(2r)}(\bar{b}) = 0$ for $r = 0, 1, \ldots$. After an antisymmetric reflection across the point $x = \bar{b}$ (or $x = \bar{a}$) is performed we obtain a smooth periodic function, which can be represented by a rapidly converging Fourier series.

Now, we can apply the pseudospectral Fourier method to the problem described by Eq. (3.5) with the "smoothed" source function $\tilde{f}$ instead of f.

$$\begin{array}{lcl} f^{(n)}(x) & \xrightarrow{\text{FFT}} & \hat{f}_k^{(n)}, \\ \hat{V}_k^{(n)} & = & -\ \hat{f}_k^{(n)}/(k^2 + \lambda^2), \\ \hat{V}_k^{(n)} & \xrightarrow{\text{FFT}} & V^{(n)}(x) \end{array}$$

The corresponding solution $\tilde{p}$ will coincide with the solution p of the original problem on the interval $x \in [a, b]$, where the projection procedure does not distort $f(x)$. Thus, solutions of (3.5), consists of the integration in the spectral space and two FFT from physical space to spectral space and vice versa.

The accuracy of the smoothing procedure, described by Eq. (3.9), depends on the number of collocation points N_ϵ in the smoothing interval of 2ϵ. If the density of points is sufficiently high, a high resolution can be obtained for a relatively small ratio ϵ/l. Thus, the relative amount of overhead due to overlapping of subdomains, needed for the smoothing operation, becomes insignificant as the number of collocation points increases.

3.4 Time-Dependent Problems

Time-dependent equations of parabolic type

$$\frac{\partial u}{\partial t} = \Delta u + N(u, u) + g(x, y, t) \qquad \text{in } \Omega \tag{3.10}$$

are in the heart of computational fluid dynamics as they model the complete Navier-Stokes system[10] or they are involved as one split computational step (together with the pressure step) in time-splitting algorithms [11]. This is also the form of the hydrodynamic equations in the vorticity/stream function formulation.

As a result of the time discretization procedure, the solution of the parabolic equation (3.10) is reduced to repeated solutions of the Helmholtz-type elliptic equation (3.5). Therefore, all DD approaches that were developed for elliptic PDE's can also be applied to solve time-dependent problems.

In a time-marching process only the RHS of (3.10) is updated in each time level, whereas the linear operator in the LHS stays invariant. This results in substantial computational savings.

One important observation concerns the structure of elliptic equations obtained from the discretization of time-dependent problems. When the time step is small, the parameter λ in (3.5) turns out to be large. This has a decisive impact on the properties of the elliptic operator in (3.5) and on the corresponding matrix after the problem is discretized in space. In particular, the boundary Green's functions are well localized near the interfaces; the same is true for the solutions, generated by a singular source function in the RHS. They decay rapidly away from the point of singularity.

Such local behaviour of the fundamental solutions simplify the solution and reflects the inherent locality of the "paternal" parabolic equation.

Recently, in [1, 4, 12], several multidomain methods were developed which take advantage, in an explicit way, of the locality of the parabolic equations in order to reduce the amount of data transfer and communication. These issues, their implementation on massively parallel machine and detailed performance analysis are the subjects of the rest of the paper.

4 Parabolic Domain Decomposition (PDD) Method

This spectral multidomain algorithm has been introduced in [1, 4, 12]. By using the local properties of the Helmholtz operator the global matching relations, where all subdomains are involved in the data exchange, become uncoupled and only local communication between contiguous subdomains remains necessary.

The application of the PDD approach along with the LFB was shown to be very efficient when applied to a 2-dimensional domain splitted into rectangular cells[12]. In that case, *all* matching relations become completely uncoupled (at the price of some overlapping of subdomains required by the LFB implementation). Thus, all communication is reduced to interactions between neighbouring elements.

4.1 The Matching Procedure

The algorithm described above can be performed concurrently in all subdomains (having S of them). The local solutions $U_p^{(n)}$ and their first derivatives will have, in general, jumps on the interfaces. To match those solutions we use a proper combination of two boundary Green's functions $h_L = e^{-\lambda x}$ and $h_R = e^{\lambda(x-l)}$, which are the homogeneous solutions of Eq. (3.5).

We present a continuous solution in each subdomain as a superposition of one particular solution and two homogeneous solutions:

$$u = \bigcup_{s=1}^{S} u^{(s)}, \qquad u^{(s)} = U_p^{(s)} + A_s e^{\lambda(x-l)} + B_s e^{-\lambda x}, \;\; x \in [0, l] \tag{4.1}$$

The unknown coefficients A_s and B_s are determined so that the continuity conditions $u^{(s)} = u^{(s+1)}$, $(u^{(s)})' = (u^{(s+1)})'$ on the interfaces $x = d_s = sl$ are satisfied.

Thus, we get a system of matching relations:

$$A_s = A_{s+1} e^{-\lambda l} - \alpha_s, \quad B_{s+1} = B_s e^{-\lambda l} - \beta_s$$

$$\alpha_s = \frac{1}{2}\left(\frac{\delta_s'}{\lambda} + \delta_s\right), \quad \beta_s = \frac{1}{2}\left(\frac{\delta_s'}{\lambda} - \delta_s\right),$$

$$\delta_s = U_p^{(s)}(l) - U_p^{(s+1)}(0), \quad \delta_s' = U_p'^{(s)}(l) - U_p'^{(s+1)}(0)$$

$$s = 1, 2, .., S-1 \tag{4.2}$$

The corresponding relations for the two extreme processors depend on the global boundary conditions. The implementation in this paper is based upon periodic boundary conditions where Dirichlet boundary condition and the complete Navier-Stokes implementation will be described in a separate paper[3].

In the case of periodic boundary condition (Eq. (3.3)) the two extreme subdomains are connected in the following way:

$$A_1 e^{-\lambda l} - A_S = \alpha_0, \qquad B_1 - B_S e^{-\lambda l} = \beta_S$$

where

$$\alpha_0 = \frac{1}{2}\left(\delta_0 + \frac{\delta_0'}{\lambda}\right), \qquad \beta_S = \frac{1}{2}\left(\delta_S - \frac{\delta_S'}{\lambda}\right),$$

$$\begin{aligned} \delta_0 = \delta_S &= U_p^{(S)}(l) - U_p^{(1)}(0) \\ \delta_0' = \delta_S' &= \tfrac{dU_p^{(S)}(l)}{dx} - \tfrac{dU_p^{(1)}(0)}{dx} \end{aligned} \tag{4.3}$$

By solving the system (4.1) we get the following expressions for the coefficients B_1 and A_S:

$$B_1 = \frac{\omega_S}{1-\omega_S}\sum_{n=1}^{S}\frac{\beta_n}{\omega_n}$$

periodic

$$A_S = -\frac{1}{1-\omega_S}\sum_{n=0}^{S-1}\alpha_n\omega_n \tag{4.4}$$

where $\omega_n = e^{-n\lambda l}$. In this case, both sets of coefficients A_n and B_n become mutually uncoupled.

The computation of the matching parameters A_n, B_n in each subdomain, using the relations (4.4), is linked to data from all the interfaces. Thus, the matching of the local particular solutions requires global communication among all subdomains (processors).

The communication among the processors remains global, because all the interfaces are involved in the matching relations. The decisive simplification, towards local communication, can be made by the following observation. Our auxiliary matching functions $e^{-\lambda x}$ and $e^{\lambda(x-l)}$ decay exponentially away from the interfaces. If the parameter λ is large enough (the time step τ of a time-marching scheme is small, respectively - see section 3.4), then these functions are well localized near the interfaces, and all the matching relations (4.2) become uncoupled. In other words, instead of a global interaction among all the subdomains, only local communication between contiguous subdomains is important. This is the main idea of the Parabolic Domain Decomposition (PDD) approach.

4.2 Extensions to Two Dimensions

Let us consider the problems (3.5) and (3.6) with periodic boundary conditions on the domain $\Omega = (0, L_x)\times(0, L_y)$, that is decomposed into parallel *strips* $\Omega = \bigcup_{n=1}^{P}\omega_n$, $\omega_n = (d_n, d_{n+1})\times(0, L_y)$, where $d_n = nl$, $n = 1, .., P$ (**case S**) or rectangular *cells* $\Omega = \bigcup_{n=1}^{P_x}\bigcup_{m=1}^{P_y}\omega_{nm}$ of the size $l_x \times l_y$; $l_x = L_x/P_x$, $l_y = L_y/P_y$ (**case C**).

Case S - Suppose that the source function $f(x, y)$ is periodic in the y direction and apply the FFT along the strips. Then, the problem is reduced to a set of one-dimensional problems for the Fourier coefficients $\hat{u}_k^{(n)}(x)$ of the type in Eq. (3.5):

$$\hat{u}_k^{(n)''} - (k^2 + \lambda^2)\hat{u}_k^{(n)} = \hat{f}_k^{(n)}(x) \qquad \text{in } \omega_n, \qquad k = -\frac{N_y}{2}, ..., \frac{N_y}{2} \tag{4.5}$$

(N_y is the number of collocation points in the y direction), which can be solved by using the algorithm with the local matching on the interfaces (described in section 4.1).

If the function $f(x,y)$ is not periodic along the strips, as it was assumed before, it can be transformed to a periodic form by using the same "folding" procedure (described in section 3.3). Then, the previous situation is reproduced on an extended region $\bar{\Omega} = (0, L_x) \times (-\epsilon, L_y + \epsilon)$. The solution of this modified problem will coincide with the original one on Ω.

Case C - We showed in [12] that the implementation of the LFB technique enables us to reduce the overall coupling of the interface points to a sequence of simple, one dimensional local matching relations. The idea is to apply this algorithm in the x and y directions separately.

In the first step we take a horizontal row of cells and transform the source function using the LFB in the y-direction. (The procedure includes an ϵ-extension and smoothing across the horizontal interfaces, anti-symmetrical reflection and an FFT on the local period).

In the second step we solve the system of one-dimensional problems of type (4.5) for the Fourier coefficients with the local matching on the vertical interfaces and then transform back into the physical space. The resulting particular solution $u^{(I)}$ will be smooth in each horizontal band except for the horizontal interfaces.

To remove discontinuities from the horizontal interfaces we transform the function $u^{(I)}(x,y)$ using the LFB in the x-direction in each vertical column and match Fourier coefficients on these lines. Then, after the application of the inverse transform into the physical space, we get a vertically smooth solution $u^{(II)}$.

Smoothing in the $y-$direction introduces discontinuities on the vertical interfaces because each Fourier coefficient is matched independently. These jumps can be removed by another matching (in the $x-$direction). Then, this is done in each new cycle along x and y alternatively. This results in a smoother solution.

5 Strip Computational Algorithm

In this section a detailed description of the actual computation of the model problem, in a strip domain decomposition form, is presented.

Initialization -

- Create the bell function (Eqs. (3.7) and (3.8)).
- Compute with initial conditions in time$= 0$.
- Compute the non-linear term term $\mathbf{N}(\mathbf{v})$ of Eq. (3.2).

Discretization (splitting) in time - The time discretization is performed by using a stiffly stable scheme investigated in [8] as was explained in section 3.2 Eq. (3.6).

Discretization (splitting) in space (Local Fourier Bases) - For space discretization the pseudospectral method is used with the local trigonometric bases as described in section 3.3.

Apply LFB on Helmholtz equation - As a result of the time discretization procedure, the solution of the parabolic equation (3.1) is reduced to the repeated solutions of Helmholtz-type elliptic equation.

Computation of the RHS and transfer of overlapped subdomains - Given

$$\nabla^2 u - \lambda^2 u = f(x,y). \tag{5.1}$$

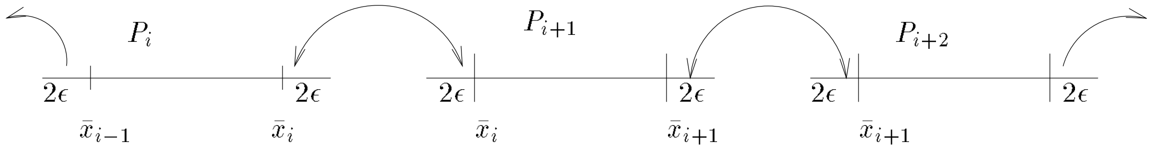

Figure 2: Transfer of overlapped contiguous subdomains between neighboring processors

If λ is large enough we achieve locality in the behaviour of the algorithm.
We compute the RHS $f(x, y)$. We solve for particular solution in each subdomain with any unspecified boundary conditions since they are unknown on the boundary as explained in section 4.1. This is accomplished via the application of the LFB method. We prepare first the overlapping in each domain that is being transferred between neighboring processors as demonstrated in Figure 2.
Continuous extension of the local pieces $f^{(n)}(x)$ of the source function beyond the interfaces

$$[\bar{x}_{n-1}, \bar{x}] \longrightarrow [\bar{x}_{n-1} - 2\epsilon_n, \bar{x} + 2\epsilon_n]$$

by communicating the neighbouring subdomains.

Projection - Projection of the local functions $f^{(n)}(x)$ by applying the procedure described in 3.3 on the extended intervals $[\bar{x}_{n-1} - 2\epsilon_n, \bar{x} + 2\epsilon_n]$. This step results in periodic continuous functions $\tilde{f}^{(n)}$.
Solution of the elemental subproblems (Eq. (5.1)) with the smoothed source functions $\tilde{f}^{(n)}$ in the RHS by means of the pseudospectral technique. It consists of the integration in the spectral space and two Fast Fourier Transforms (FFT) from the physical to the spectral space and vice versa. In our case, we use instead the discrete cosine transform (DCT) as explained in section 8. We apply the smoothing algorithm to the pieces of the source function $f^{(n)}(x)$. Now, equation (5.1), with the function $\tilde{f}$ in the RHS, can be solved with spectral accuracy (as explained in section 3.3).
The smoothing procedure does not change the primary source function *inside* the subdomains $[\bar{x}_{n-1}, \bar{x}]$ but only distorts the extra pieces on the *external* intervals $[\bar{x}_{n-1} - \epsilon_n, \bar{x}_{n-1}]$ and $[\bar{x}_n, \bar{x} + \epsilon_n]$. Therefore, within each subdomain the solution $\tilde{u}^{(n)}(x)$ of the "smoothed" problem obviously coincides with the particular solution $u_p^{(n)}(x)$ of the original problem (5.1). Hence, our approach does not require any "unfolding" operations to recover the original non-periodic solution. (The unfolding is the inverse operation of (3.9) associated with the LFB construction which recovers the original function from its folding operation). The only extra work is the overlapping which have to be done in order to replace the solution of the non-periodic local problems in subdomains by the solution of the smooth periodic problems using the spectral Fourier method.

Overlapping of adjacent subdomains - over the distance $2\epsilon_n$ by means of local communication between neighbouring processors.

FFT on a larger period - Perform the FFT on the larger period $(\bar{x}_{n-1} - \epsilon_n, \bar{x}_n + \epsilon_n)$ instead of $(\bar{x}_{n-1}, \bar{x})$.

Smoothing - Extra operations (3.9) for smoothing the local functions. The smoothing is done as was explained in section 3.3 and figure 1.

It was shown in [1] that the only factor which affects the precision of the smoothing procedure is the number of collocation points n_ϵ in the range of smoothing 2ϵ (scaling reasoning). Consequently, the relative amount of overhead due to to overlapping decreases as n_ϵ/n (here n is the number of collocation points in a subdomain) so that it turns to be insignificant for large enough problems.

6 Cell Computational Algorithm

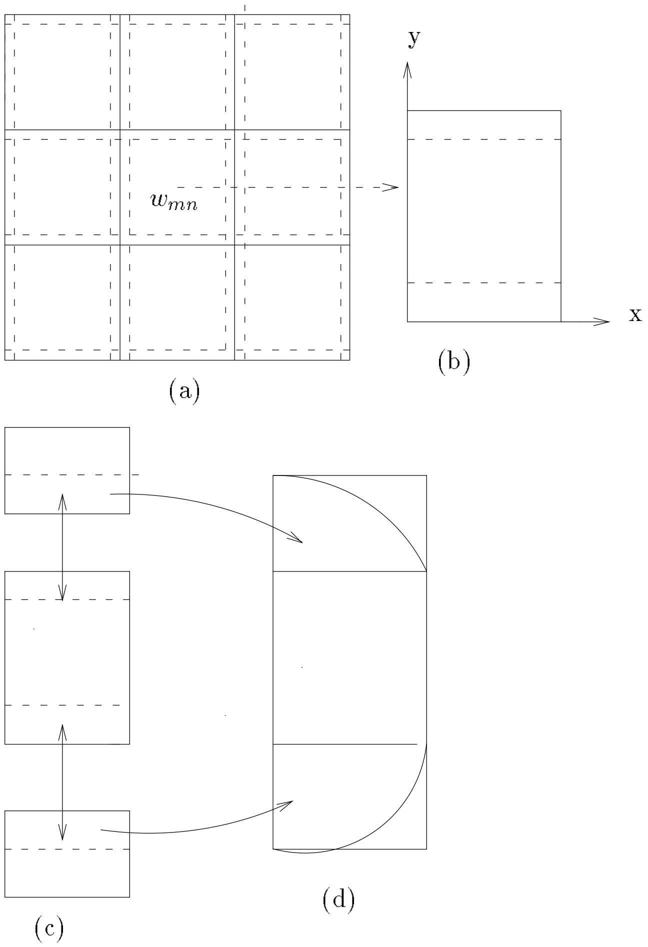

Figure 3: Implementation of the cell algorithm: (a) The domain is divided into $P_x \times P_y$ processors. (b) The overlapped area of each cell in one direction marked by the dotted lines. (c) and (d) show the extension and smoothing of the local solution.

We solve the two-dimensional Helmholtz equation (5.1) in the rectangular cells $\Omega = \bigcup_{n=1}^{P_x} \bigcup_{m=1}^{P_y} \omega_{mn}$ (figure 3(a)) of sizes $l_x \times l_y$; $l_x = L_x/P_x$, $l_y = L_y/P_y$ (see Fig. 3(a)). P_x and P_y are the number of processors along the x and y axis respectively.

Instead of *simultaneous* matching at all collocation points we rather match *successively* in x and y directions.

1. We take horizontal row of cells $\Omega_\nu = \bigcup_{m=1}^{P_x} \omega_{\nu m}$, $\nu = 1, \ldots, P_y$ and transform the source function $f(x, y)$ into the LFB in the y-direction (the procedure includes an ϵ-extension, smoothing

across the boundaries $\bar{y} = (\nu - 1)l_y - \epsilon$ and $\bar{y} = \nu l_y + \epsilon$, antisymmetric reflection and FFT on the local period $\tau = 2(l_y + 2\epsilon)$- see section 5 and figure 3 (c) and (d)).

2. We solve the system of one-dimensional problems of the type

$$\frac{d^2\hat{U}_k^{(n)}}{dx^2} - (k^2+\lambda^2)\hat{U}_k^{(n)} = \hat{f}_k^{(n)}(x) \qquad \text{in } \omega_n, \qquad k = -\frac{N_y}{2}, ..., \frac{N_y}{2}$$

   (N_y is the number of collocation points in the y direction) for the Fourier coefficients with the local matching on the vertical interfaces, after which we transform back into the physical space.

3. The resulting particular solution $U_p(x, y)$ will be smooth in each horizontal band except for the interfaces $\bar{y}_n = nl_y,\ n = 1, \ldots, P_x$ (figure 3(c,d)).

4. To remove discontinuities from the horizontal interfaces we use again our one-dimensional matching routine. We take the vertical column $\Omega^\mu = \bigcup_{n=1}^{P_y} \omega_{n\mu},\ \mu = 1, \ldots, P_x$, transform U_p into the LFB (by analogy with the source function on the first step) and implement the matching on the lines $\bar{y}_n$ for each Fourier coefficient. By inverting the transform into the physical space we get the vertically smooth solution ${U^I}_p$.

   Smoothing in the y direction induces discontinuities on the vertical lines $\bar{x}_n = nl_x,\ n = 1, \ldots, P_x$, because each Fourier coefficient is matched independently. These jumps are removed by another matching (in the x direction). Each new "ironing" along x and y alternatively results in a smoother solution.

The proposed algorithm is an extension (generalization) of the strip computational algorithm since it can be done in the following way:

1. Do steps (a)-(d) as described in figure 3. In other words, as in the LFB, we take extension and perform smoothing of the local particular solution U_p..
2. As in the strip computation, we match in the x direction.
3. Extension, smoothing and FFT as explained in the strip computation.
4. As in the strip computation, we match in the y direction.
5. Go to step 2.

7 Implementation

The proposed parallel algorithms are appropriate for the implementation on distributed memory computers. Currently, the code is ported onto the distributed memory Meiko computer (MK096). Meiko's i860 processing nodes combine the high performance of the Intel i860 processor with the advanced communication of the Inmos transputers. Each MK096 element comprises a 40 MHz Intel i860 with scalable DRAM 8 MB (only two processors have 16MB - for serial measurements) and sharing 2 MBytes of communication memory, and two Inmos transputers T800 providing the node with 8 communication links. The interface between the i860 and the transputers is invisible to the user - from a software perspective the i860 node behaves like an i860 processor with 8 communication links. The main feature of this computer is its *flexible network configuration*. Special software tools allow the user to create several hardware links between any pair of processors. Thus, instead of logical connections (when data usually paths through a line of transputers), one can create a certain group of directly connected processors. Flexible network topology is a powerful tool for optimization of parallel algorithms and shows to be an efficient facility to prevent bottleneck problems.

Effective implementation of the code on a distributed memory parallel computer requires the development of an efficient concurrent algorithm, which should provide minimization of communication costs and proper load balancing. The basis of the parallel implementation adopted here is the domain decomposition (DD) method as described in sections 5 and 6. Two types of domain decomposition are considered:

1. Parallel strips
2. Rectangular cells

In Figures 4 and 5 we present the computational domain, the physical connections among the processors and the processors assignments. Two different setups are considered in the computation using strips (Figure 4). The X-direction is divided into a number of overlapping subdomains in the sense that each processor (represented by a box in figure 4) has to send and receive only from his immediate neighboring processor some overlapped information according to the LFB methodology. Then, if we use topology (a), in each time step the overall communication time increases as the number of processors increases because a processor can start to process only after its two neighbors complete the transfer of their overlapped areas to each other. This will not degrade substantially the performance and will not damage the scalability nature of the algorithm (as explained later) because the communication is local and as long as the size of the problem in each processor increases the time spent on local communication is negligible. We adopt topology (b) in our strip implementation. Then, each advance in time step only depends on two neighbors instead of three as in topology (a). That means that every two neighbors does not have to wait and the overall time spent in each time slice is reduced in compare to connection (a). In the Y-direction, the domain is splitted into non-overlapping of equal size with the same number of points. Most of the time spent in the X-direction is the result of the application of the algorithm as detailed in section 5. The additional computational time along the Y-direction is linear when the size of the Y-direction increases linearly (as explained later). Therefore, the size of the Y-direction is insignificant. Since its "linearity" (from the performance point of view) his time can be predicted accurately and it is optimal.

Cell implementation, which is based on strip computation, also uses topology (b) in the X and Y-directions, respectively. In the cell computation (Figure 5), the X and the Y-directions are assigned a group of processors as a two-dimensional matrix (not necessarily square—see column one in the cell timing table 7). Cell in location i and j is assigned to a processor. The communication between neighboring processors (X and Y directions) is based on topology (b) in Figure 4 which means that we do not have to wait till all the processors complete to transfer their overlapped domains in each time slice in order to proceed.

The proposed algorithms are naturally suited for the DD conception. This approach allows a uniform memory allocation among processors and since the work per grid point is roughly equal, assigning the same number of grid points to each subdomain assures good load balancing. The original problem is easily decoupled and solved in each subdomain almost independently. Application of efficient concurrent algorithms enables to achieve the main objective of the approach - to perform most of the computational work within each subdomain and therefore to reduce communication and synchronization overheads.

The proposed implementation strategy can easily be changed to reflect another advantage of the current approach in which it allows the implementation of different resolution in each subdomain,

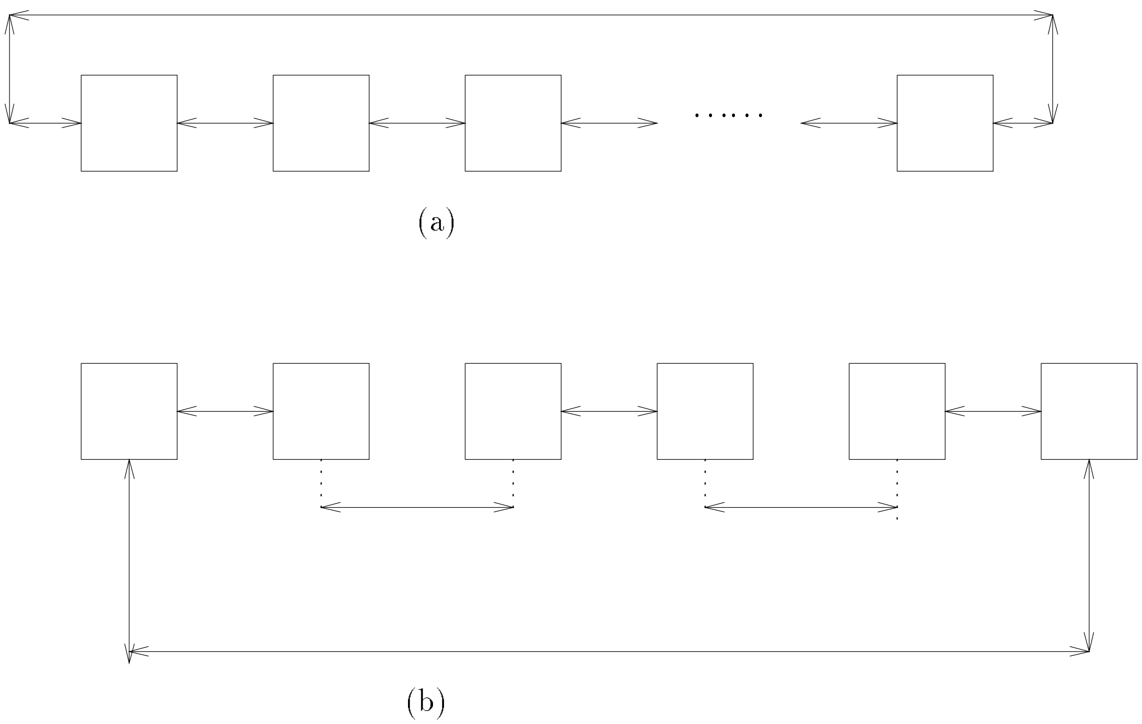

Figure 4: Two topologies for strip computation

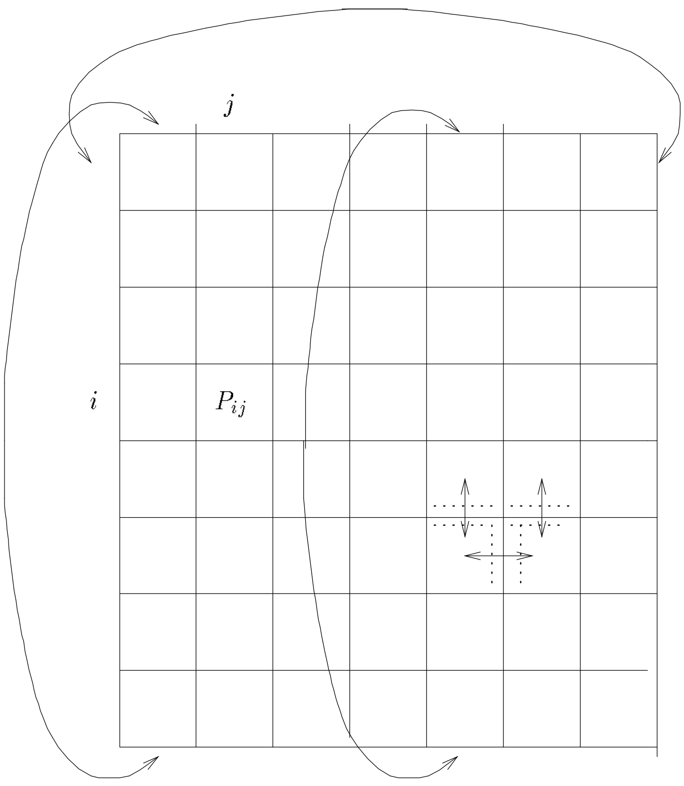

Figure 5: Cell computation

N_x	T_1	N_x	T_1	N_x	T_1
128	1.6	3200	71.1	11520	353.5
256	3.8	3840	91.6	12228	375.7
512	8.1	4096	100.0	12800	393.2
768	12.7	4608	116.6	15360	471.1
1024	17.1	5120	133.1	16384	503.0
1280	20.5	6144	165.3	19200	597.8
1536	27.8	6400	173.2	20480	638.7
1920	35.4	6912	188.9	20736	647.7
2048	37.0	7680	208.8	32040	1090.0
2304	43.0	8192	226.0	32768	1121.0
2560	51.4	9216	254.4	65536	2620.0
3072	67.0	10240	281.6		

Table 1: Serial timing (in seconds) for 10 iterations; $N_y = 8$

which makes it valuable as an adaptive algorithm [4]. We can generalize the matching procedure in [1, 12], which is valid for uniform domain decomposition, onto the case of arbitrary domain decomposition in order to have variable resolution at different locations in the physical space. This leads us to the possibility of treating unstructured grids[14].

In any case, in uniform or nonuniform grid distribution, the communication is reduced to local interactions only between neighbouring subdomains. The implementation in this paper reflects uniform domain decomposition.

8 Performance Analysis

Notations:

$N_x, \; N_y$	number of collocation points in x and y directions
$N_T = N_x N_y$	total number of collocation points in the computational region
$P = P_x \times P_y$	number of processors
$n = N_x/P_x, \; \; m = N_y/P_y$	number of points in a processor in x and y directions
N_ϵ	number of points on an overlapped interval ϵ
T_1	best serial time
T_S	time of parallel computation when the domain is decomposed into strips
T_C	time of parallel computation when the domain is decomposed into cells
$S_p = T_1/T_S \; \; (S_p = T_1/T_C)$	speed up
$E_p = S_p/P, \; \; 0 < E_p < 1$	efficiency
$N = 2(n + 2N_\epsilon)$	the size of the local FFT
T_{DCT}	parallel time in each processor using the Discrete Cosine Transform (DCT)

In order to evaluate the achieved speedup by the various algorithms serial timing has to be measured and analyzed. The dependence of the serial time T_1 (table 1) on the number of points N_x is not smooth (even not monotonic). The reason is that we are using Temperton's FFT algorithm which depends how the number N_x is factorized into its primitive multipliers: $N = 2^i \; 3^j \; 5^k$ where

i, j, k are integers. In other words, the performance of the algorithm is different for numbers which are powers of $2, 3$ or 5 only, or contains mix of different multipliers. However, if the size of the FFT is based on one factor only, say $N = 2^i$, then the dependence of T_1 on N is depicted by a smooth curve (Fig. (6)). For further estimation of the speedup, the serial timing for an arbitrary N (not necessarily based only on $2, 3$ and 5 separately) is required. We estimate it by interpolation of known values provided the dependence is smooth (as in Fig. 6).

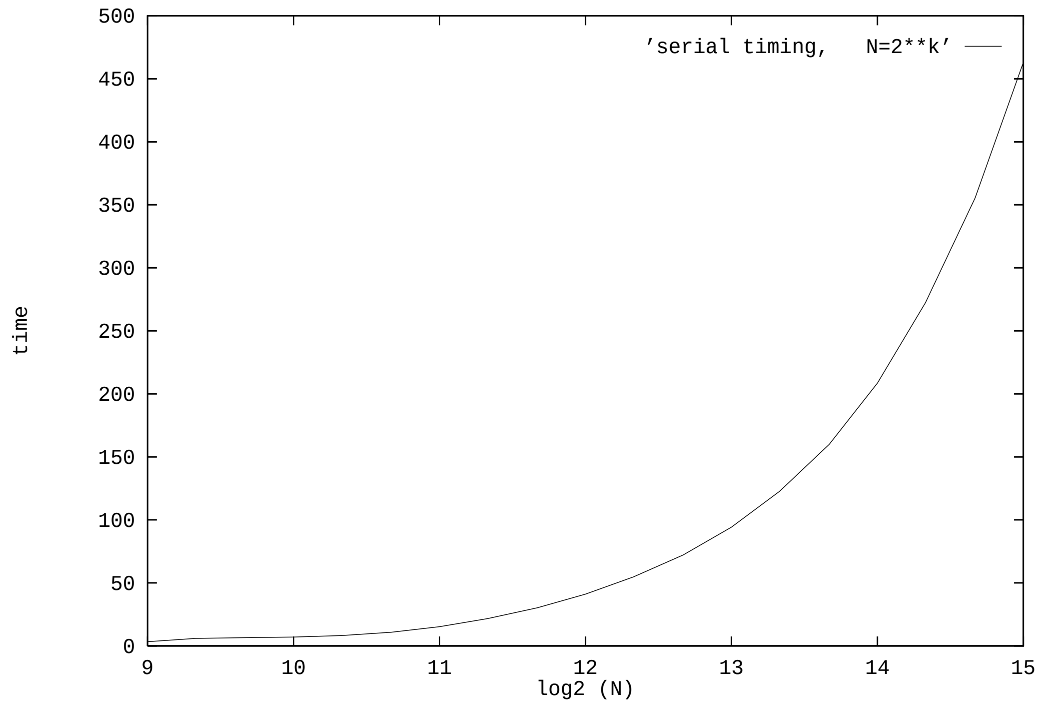

Figure 6: Serial timing (in sec.) where the size is $N = 2^k$ for 10 iterations and $N_y = 8$.

Since the FFT routine constitutes a major part of our multidomain LFB approach, the dependence $T_1(N)$ is computed, to a great extent, by the expression

$$T_1 = \mu N_x N_y (log N_x + log N_y) \tag{8.1}$$

in particular for the FFT algorithm, where N_x and N_y are the the number of points in the X and Y-directions, respectively.

In addition to the measured serial timing presented in Table 1, we also present the serial timing and the coefficient μ (Eq. (8.1)) for several values of N_y (Table 2), allowed by the limited storage of 16MB per processor. (Most of the processors in the particular configuration of the Meiko we used have 8MB of storage. Some of them have 16MB. The serial measurements were taken on the 16MB processor. The empty slots in table 2 are due to memory limitations.) The coefficient μ (Eq. (8.1)) in the "pure" FFT algorithm is independent of the total number of points $N_T = N_x N_y$ in the region. However, because of the presence of other factors in the program and also due to some particular features of the Meiko machine such as the size of the cache and others, this factor depends slightly on N_x and N_y. Since the factor μ increases monotonically with the numbers N_x and N_y, the expression in Eq. (8.1) and the data in Table 2 are used to estimate the minimal serial time and the *least* speedup respectively.

N_x	$N_y = 16$		$N_y = 32$		$N_y = 64$	
	T	$\mu \cdot 10^4$	T	$\mu \cdot 10^4$	T	$\mu \cdot 10^4$
128	3.4	2.2	7.7	2.2	16.2	2.2
256	7.8	2.2	17.4	2.2	36.2	2.2
512	16.7	2.2	37.1	2.2	77.9	2.2
1024	35.2	2.2	79.5	2.2	176.9	2.2
2048	78.2	2.2	184.2	2.4	390.6	2.4
4096	216.4	2.8	485.4	2.4	1001.8	2.4
8192	488.3	3.0	-	> 3.0	-	> 3.0
N_x	$N_y = 128$		$N_y = 256$		$N_y = 512$	
	T	$\mu \cdot 10^4$	T	$\mu \cdot 10^4$	T	$\mu \cdot 10^4$
128	34.8	2.2	80	2.2	164	2.2
256	77.2	2.2	171	2.2	368	2.2
512	170.4	2.2	350	2.2	720	2.2
1024	372.6	2.3	760	2.3	-	> 2.3
2048	808.2	2.4	-	> 2.4	-	> 2.4
4096	2086.6	2.4				

Table 2: Serial timing (in sec.) for 10 iterations

P	$n = 32$, $N = 128$		$n = 96$, $N = 256$		$n = 224$, $N = 512$		$n = 480$, $N = 1024$		$n = 992$, $N = 2048$		$n = 2016$, $N = 4096$		$n = 4032$, $N = 8192$	
	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p
4	1.1	1.4	2.7	2.1	5.6	2.6	11.9	2.8	24.4	3.9	58.7	3.8	123.4	4.0
8	1.2	3.1	2.7	4.7	5.6	5.8	11.9	7.7	24.6	8.8	58.8	8.4	123.4	8.9
16	1.2	6.0	2.9	9.6	5.8	14.3	12.0	17.4	24.6	19.7	58.9	18.9	123.6	20.8
20	1.4	7.4	2.9	12.2	6.0	18.4	12.2	22.3	24.7	25.3	58.9	24.5	123.7	26.8
22	1.5	8.1	3.0	13.4	6.0	20.3	12.2	25.0	24.8	28.1	59.1	27.4	123.9	29.8
26	1.5	9.6	3.0	16.0	6.2	24.4	12.4	29.8	24.9	34.0	59.2	33.0	124.0	35.6

Table 3: Timing (in sec.) and speedup for strip computation (topology b) with 10 iterations; $N = 2(n + 2N_\epsilon)$, $N_x = n \cdot P_x$, $N_y = 8$, $N_\epsilon = 16$

P	$n = 32$, $N = 128$		$n = 96$, $N = 256$		$n = 224$, $N = 512$		$n = 480$, $N = 1024$		$n = 992$, $N = 2048$		$n = 2016$, $N = 4096$	
	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p
4	2.1	1.6	5.5	2.2	12.0	2.6	25.4	2.8	53.4	3.9	126.9	3.8
8	2.3	3.4	5.7	4.6	12.2	6.0	25.6	7.7	53.6	8.8	127.0	8.1
16	2.4	7.0	5.9	8.8	12.4	14.5	25.9	17.6	53.9	18.9	127.2	18.2
20	2.5	8.5	6.0	11.8	12.5	19.5	26.0	21.9	54.1	23.7	127.3	24.4
22	2.7	8.8	6.1	13.4	12.8	21.5	26.1	23.9	54.3	26.2	127.5	26.9
26	2.9	9.8	6.3	16.8	12.9	26.1	26.2	27.6	54.6	31.2	127.6	32.3

Table 4: Timing (in sec.) and speedup for strip computation (topology b) with 10 iterations; $N = 2(n + 2N_\epsilon)$, $N_x = n \cdot P_x$, $N_y = 16$, $N_\epsilon = 16$

P	$n = 32$, $N = 128$		$n = 96$, $N = 256$		$n = 224$, $N = 512$		$n = 480$, $N = 1024$		$n = 992$, $N = 2048$		$n = 2016$, $N = 4096$	
	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p
4	4.4	1.7	11.3	2.4	24.8	2.8	53.1	6.6	112.5	4.1	263.6	3.9
8	4.5	3.9	11.6	5.0	25.0	6.1	53.2	8.4	112.8	9.6	263.9	8.8
16	4.8	7.7	11.7	10.6	25.2	16.1	53.4	19.5	112.9	20.1	264.1	19.5
20	4.9	9.7	11.8	14.2	25.4	21.5	53.5	23.2	113.0	25.5	264.4	24.8
22	4.9	10.8	12.0	16.0	25.4	24.2	53.7	25.6	113.2	28.2	264.5	27.4
26	5.1	12.5	12.2	19.3	25.6	28.4	53.8	30.6	113.3	33.7	264.7	34.5

Table 5: Timing (in sec.) and speedup for strip computation (topology b) with 10 iterations; $N = 2(n + 2N_\epsilon)$, $N_x = n \cdot P_x$, $N_y = 32$, $N_\epsilon = 16$

P	$n = 32$, $N = 128$		$n = 96$, $N = 256$		$n = 224$, $N = 512$		$n = 480$, $N = 1024$		$n = 992$, $N = 2048$	
	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p	T_{DCT}	S_p
4	8.9	1.8	22.8	2.4	50.1	2.8	107.2	3.3	239.0	4.0
8	9.1	4.0	22.8	5.6	50.2	6.5	107.4	8.5	239.2	9.3
16	9.2	8.5	22.9	11.8	50.4	16.6	107.5	19.9	239.5	19.9
20	9.4	10.8	23.1	15.5	50.4	22.3	107.8	25.8	239.6	26.8
22	9.6	11.9	23.3	17.5	50.7	25.0	107.9	28.5	239.7	29.6
26	9.7	14.3	23.4	21.9	50.8	30.6	107.9	34.2	239.8	35.4

Table 6: Timing (in sec.) and speedup for strip computation (topology b) with 10 iterations; $N = 2(n + 2N_\epsilon)$, $N_x = n \cdot P_x$, $N_y = 64$, $N_\epsilon = 16$

$P = P_x \times P_y$	$N = 128$ $n = m = 32$			$N = 256$ $n = m = 96$		
	T_{DCT}	S_p	E_p	T_{DCT}	S_p	E_p
$4 = 2 \times 2$	9.3	0.8	0.2	51.2	1.7	0.4
$6 = 2 \times 3$	10.3	1.1	0.2	51.4	2.6	0.4
$8 = 2 \times 4$	10.5	1.5	0.2	51.4	3.5	0.4
$9 = 3 \times 3$	10.7	1.7	0.2	51.1	4.0	0.4
$12 = 3 \times 4$	10.5	2.4	0.2	52.1	5.4	0.5
$16 = 4 \times 4$	10.2	3.4	0.2	52.4	7.4	0.5
$20 = 4 \times 5$	10.6	4.2	0.2	52.6	9.3	0.5
$25 = 5 \times 5$	10.8	5.3	0.2	52.8	11.9	0.5

Table 7: Timing (in sec.) and speedup for cell computation (topology b) with 10 iteration; $N_\epsilon = 16$

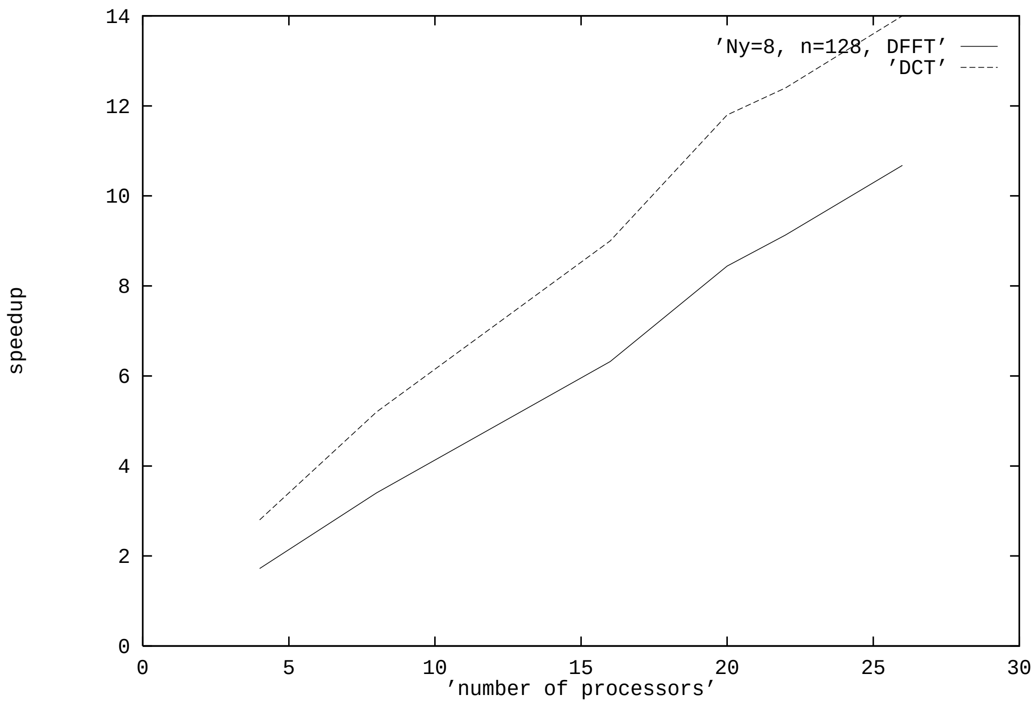

Figure 7: The different speedups achieved for strip computation when the FFT and DCT are used for $N_y = 8$.

We can utilize the fact that the the folding procedure, as was explained in section 3.3, is based on the symmetry of the bell functions. Thus, the discrete cosine transform (DCT) should be used rather than the FFT. It saves about $40 - 50\%$ of CPU time of the *transform* operations. Figure 7 demonstrates the advantage of using the DCT over the use of the FFT. From now on, the DCT is employed instead of using the FFT in the pseudospectral phase of the algorithm. Therefore, in tables 3-7 the total time of the algorithm in each processor is denoted by T_{DCT} to emphasize the fact that the kernel of the pseudospectral computation is based on the DCT.

From the performance analysis we can see:

Scalability - Meiko is a message-passing MIMD multiprocessor which in principle is able to grow in accordance with the requirements of the task. The results show that our algorithm is scalable in the sense that problems of equal size uses the same time when the number of processors increase because the communication is reduced from global to local and thus the solution depends on direct neighboring processors.

To demonstrate the scalability we have to look at the columns labeled T_{DCT} in tables 3-7. For example, in table 3 we present the results of running the strip algorithm for seven different problem sizes where $N_y = 8$. In column one, the difference between the T_{DCT} for 4 processors and 26 processors is 36%. As the problem size increases this difference decreases as it is shown in table 8. For the largest problem size, the difference in the T_{DCT} between 4 and 26 processors is 1%.

$n = 32$ $N = 128$	$n = 96$ $N = 256$	$n = 224$ $N = 512$	$n = 480$ $N = 1024$	$n = 992$ $N = 2048$	$n = 2016$ $N = 4096$	$n = 4032$ $N = 8192$
36%	11%	11%	5%	2%	1%	1%

Table 8: Difference between 4 processors and 26 processors in % of the T_{DCT} time for each problem size in table 3

We see that for larger problems as the number of the processors increases the time stays almost the same. As explained in the section 7, neighboring processors are connected to each other using topology b (figures 4 and 5). Then, as the number of processors increases the communication time increases also. The small change in the T_{DCT} is unavoidable. But it indicates very clearly that the algorithm is scalable. When the ratio of computation/communication increases (the size of the problem in each domain increases) then the scalability phenomenon is even clearer.

When N_y is increased by a factor of two then we can see that the T_{DCT} time in table 4 is almost twice the comparable T_{DCT} in table 3. When we compare the T_{DCT} measurements in tables 3, 4, 5 and 6 we see that when N_y increase by a factor of two its T_{DCT} increases by the same factor. The difference between column one and the last column in table 4 is the same as appeared in table 8 for table 3. Therefore, even for larger N_y the strip algorithm is scalable.

For cell computation we see from table 7 that the same phenomenon appears. The time difference in the first column of T_{DCT} between 4 and 25 processors is 16% while in the last column (where the problem size is the biggest) the difference between 4 and 25 processors is 3.0%. Here we present only problems of very modest sizes because of the storage limitation of the current configuration (8MB per processor).

The boundary Green's functions are used as the connection between the elemental solutions in subdomains. We show that for *parabolic problems* only *local* communications between the processors links are required, so that the parallel algorithm is *fully scalable*.

Speedup - Speedup grows with the increase of problem size as it demonstrated in Figures 8 and 9.

Table 3 presents the computational time and the achieved speedup from using *strip* type computation. The results in table 3 are presented for $N_y = 8$. Almost the same speedup is achieved for *any* other values of $N_y = 16, 32, 64$ (tables 4,5, 6, respectively) as the computational time obeys the dependence $T \propto N_x N_y log N_y$ for both the serial and the strip implementation. Some additional factors, appearing in the parallel computation, may affect the speedup. One of these is the increase of transfered data when the number of points in the $y-$direction increases. However, the speed of the Meiko internal network is about 1 MB per second so that this factor slightly affects the CPU time for those numbers N_x, N_y which are of practical interest. Another factor is how the *local* size of the FFT is decomposed into its primitive multipliers $N = 2^i 3^j 5^k$.

The speedup in the strip computation is almost linear as the size of the problem increases. The same speedup is achieved if N_y increases. As was explained before, most of the work is performed along the x-direction.

Therefore, there is almost a linear time increase between T_{DCT} in table 3 and tables 4-6 where in each table N_y is multiplied by 2.

We can see that when the problem size is big enough and it fits the size of the cache we get super-linear speedup as it appears in table 3 for the columns starting at $n = 480, N = 1024$, table 4 for the columns starting at $n = 480, N = 1024$ and tables 5 and 6 for the columns starting at $n = 224, N = 512$.

When the computation is performed using the *cell* strategy the size of the storage in each processor dictates the size of the experiment. Only $n = 32, 96$ can fit into the machine. Figure 9 presents the achieved linear speedup for $P \leq 25$. We can see that the speedup increases as the size of the problem in each processor increases. For $P > 25$, the S_p is calculated as follows: the executable time is assumed invariant when P increases due to the scalability of the algorithm. The serial time T_1 is estimated using the expression (8.1) and the values of μ are derived from Table 1. The upper line ($n = 256$ in figure 9) is plotted by using the extrapolated values for S_p from different n at each particular P.

Efficiency - We see from table 3 and figure 10 that when using strip computation for large enough n and P the efficiency $E_p = S_p/P$ is approaching 1.

The efficiency for the cell computation, as presented in table 7, is not high and achieves only about 20 and 40%. Due to storage limitation only modest size problems were fitted into each processor. However, the trend is such that E_p grows (almost *linearly*) with the number of points n inside each processor.

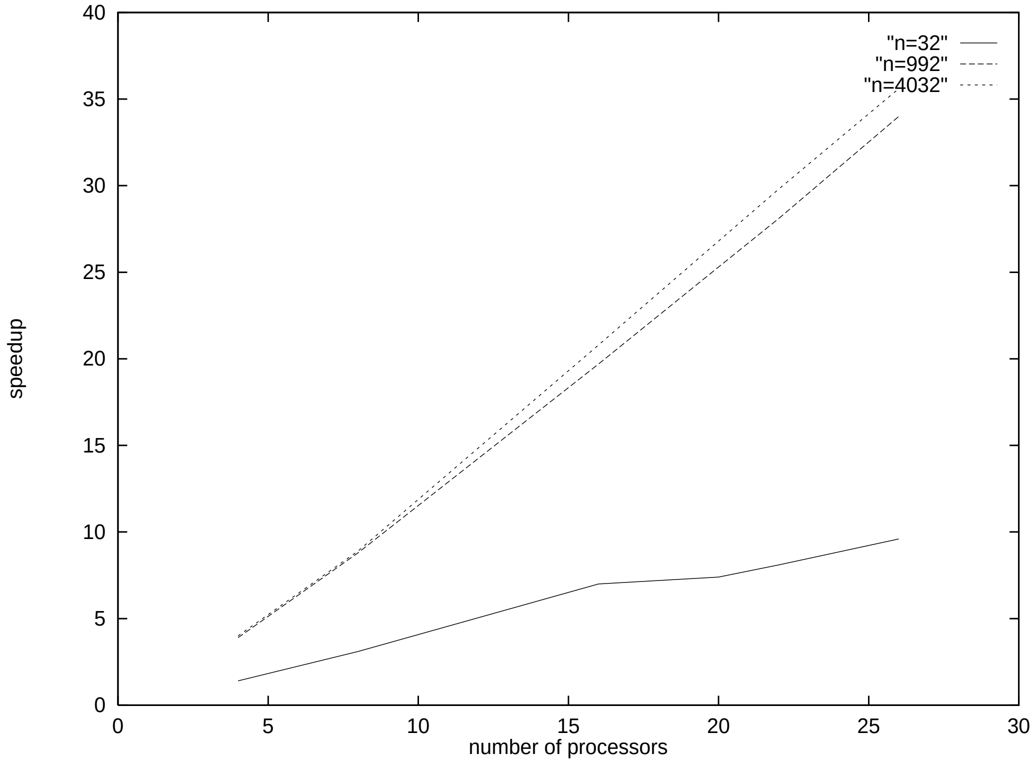

Figure 8: Achieved speedup when the domain decomposed into strips, $N_y = 8$.

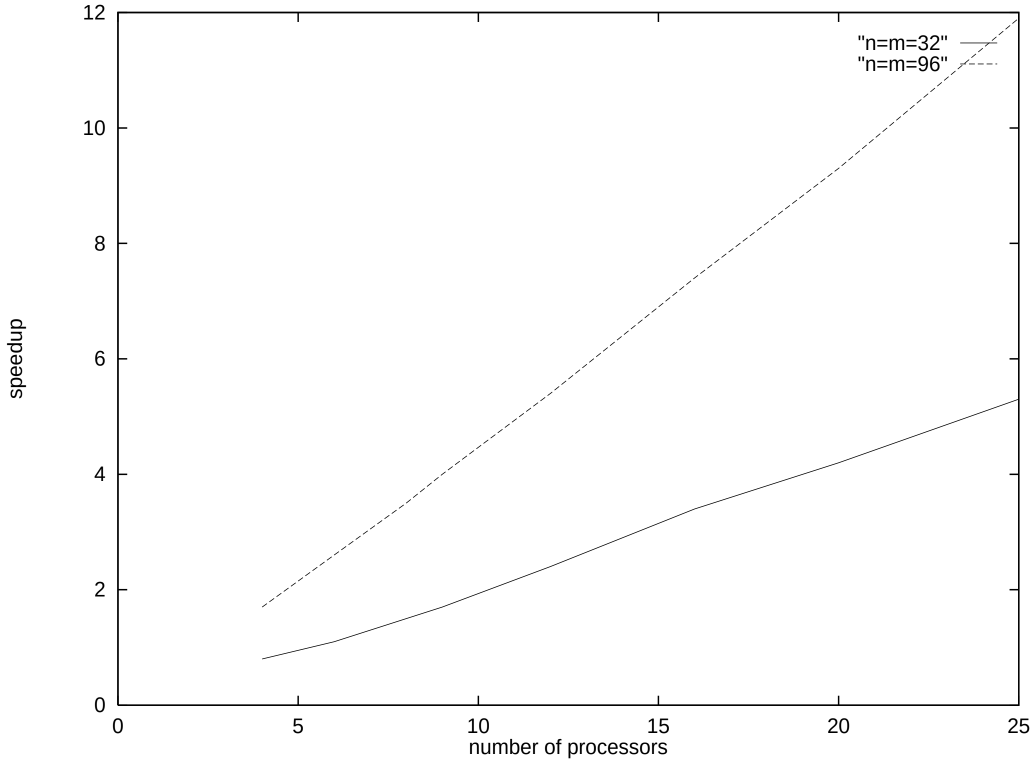

Figure 9: Achieved speedup when the domain decomposed into cells.

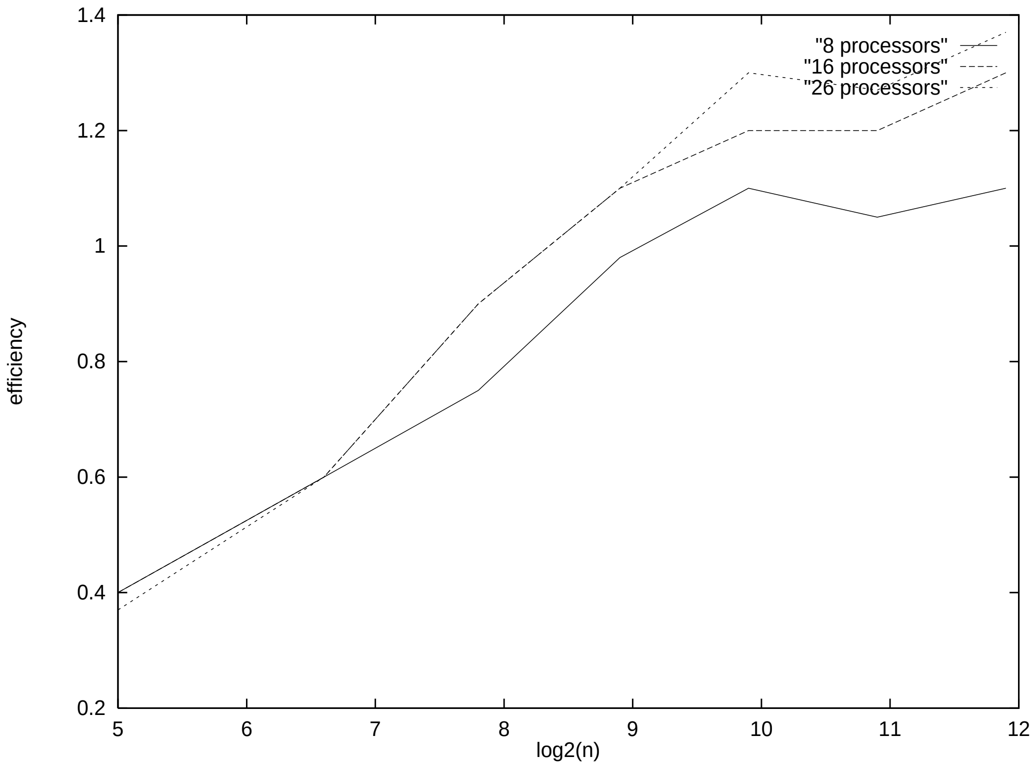

Figure 10: Efficiency when the domain decomposed into strips and $N_y = 8$ based upon table 3

9 The ADI Method

The alternate direction implicit (ADI) method is another efficient time-discretization scheme. In this section we describe how the ADI method was implemented for the domain decomposition in the form of strips as well as cells. The performance analysis of this implementation related to cells is described.

9.1 Discretization in Time

An efficient time-discretization scheme can be constructed by using the alternate direction implicit (ADI) method. This is a two-step method which can be used as

$$\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t/2} = \nu(\mathbf{v}^*_{xx} + \mathbf{v}^n_{yy}) - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{9.1}$$

$$\frac{\mathbf{v}^{n+1} - \mathbf{v}^*}{\Delta t/2} = \nu(\mathbf{v}^*_{xx} + \mathbf{v}^{n+1}_{yy}) - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{9.2}$$

When the $2-d$ order Adams-Bashford scheme is employed for the approximation of the non-linear and the non-homogeneous terms, the following parameters are prescribed:
$J_e = 2,\ \beta_0 = 1.5,\ \beta_1 = 0.5.$

The ADI method allows us to split a 2-D problem into two *ordinary* 1-D equations such that the first one (9.1) is solved in the x direction, and the second equation (9.2) is solved in the y direction. The main advantage of this approach is that it enables to perform the time-advancing step in the *physical* space without preliminary transform into the spectral space in the transverse direction (as was done in the previous sections).

We rewrite the system (9.1)-(9.2) in the form

$$\mathbf{v}^*_{xx} - \lambda^2 \mathbf{v}^* = -\lambda^2 \mathbf{v}^n - \mathbf{v}^n_{yy} + \frac{1}{\nu}\mathbf{G} \equiv \mathcal{R}_1 \tag{9.3}$$

$$\mathbf{v}^{n+1}_{yy} - \lambda^2 \mathbf{v}^{n+1} = -\lambda^2 \mathbf{v}^* - \mathbf{v}^*_{xx} + \frac{1}{\nu}\mathbf{G} \equiv \mathcal{R}_2 \tag{9.4}$$

where

$$\lambda^2 = \frac{2}{\nu \Delta t},$$

$$\mathbf{G} = \frac{3}{2}(\mathbf{N} - \mathbf{F})^n - \frac{1}{2}(\mathbf{N} - \mathbf{F})^{n-1}$$

The most precise and efficient way for the evaluation of non-homogeneous terms in the RHS of (9.3)-(9.4), which does not require any additional calculation of derivatives, is done in the following way:

$$\mathcal{R}_1 = -\mathcal{R}_2 - 2\lambda^2 \mathbf{v}^{n+1} + \mathbf{G}/\nu, \tag{9.5}$$

$$\mathcal{R}_2 = -\mathcal{R}_1 - 2\lambda^2 \mathbf{v}^* + \mathbf{G}/\nu \tag{9.6}$$

Equation (9.3) is solved for $\mathbf{v}^*(x_i, y_j)$ at each $y_j = j\Delta y,\ \Delta y = 2\pi/N_y,\ j = 0, 1, ..., N_y$ and Eq. (9.4) is solved for $\mathbf{v}^{n+1}$ at each $x_i = i\Delta x,\ \Delta x = 2\pi/N_x,\ i = 0, 1, ..., N_x$ by using the LFB method.

The ADI method can be implemented for domain decomposition in the form of cells as well as strips. In the case of strips, the matching of the particular solutions is performed for Eq. (9.5) only on the vertical interfaces $\bar{x}_q = q\ l_x,\quad l_x = 2\pi/P_x, /q = 1, 2, ..., P_x$, while Eq. (9.4) is solved with the explicit (periodic) boundary conditions. In the case of cells, the matching is performed both on the vertical (for Eq. (9.3)) and the horizontal interfaces $\bar{y}_r = r\ l_y,\quad r = 1, 2, ..., P_y,\ l_y = 2\pi/P_y$ for Eq. (9.4).

The general ADI computational algorithm utilizing domain decomposition in the form of cells is identical to one described in section 4.2 but the time-advancing step is performed in physical space without transforming into spectral space. Therefore, the algorithm is very close to the one described in section 6 with the above differences.

9.2 ADI Performance Analysis

The same type of analysis that was done in section 8 for cell computation (include the severe limitation of storage in each processor) and its conclusions are also applicable to ADI performance analysis. We should emphasize again here that the sizes of the problem fitted into each processor is very small and extrapolated based results are needed to demonstrate the best performance of the algorithm.

The T_{DCT} values in table 9 show that our parallel algorithm is fully scalable. We see that as the number of the processors increases the time stays almost the same. Even the difference between 4 processors and 25 processors for the smallest size problem is again 15%.

Speedup grows linearly with the number of processors P as it demonstrated in Figure 11.

$P = P_x \times P_y$	$n=m=32$			$n=m=96$			$n=m=128$			$n=m=224$		
	T_{DCT}	S_p	E_p	T_{DCT}	S_p	E_p	T_{DCT}	S_p	E_p	T_{DCT}	S_p	E_p
$4 = 2 \times 2$	14	0.58	0.14	92	1.0	0.25	166	1.04	0.26	468	1.3	0.32
$6 = 2 \times 3$	15	0.85	0.14	95	1.5	0.25	169	1.6	0.26	475	2.0	0.33
$8 = 2 \times 4$	14	1.26	0.16	94	2.1	0.26	169	2.2	0.27	471	2.7	0.34
$9 = 3 \times 3$	16	1.25	0.14	97	2.3	0.26	175	2.4	0.27	478	3.0	0.33
$16 = 4 \times 4$	14	2.7	0.17	97	4.3	0.27	185	4.4	0.27	482	5.6	0.35
$20 = 4 \times 5$	18	2.7	0.13	112	4.8	0.24	179	5.8	0.29	489	7.1	0.35
$25 = 5 \times 5$	16	3.8	0.16	102	6.9	0.28	180	7.3	0.30	475	9.1	0.36
$30 = 5 \times 6$	-"-	4.7	0.16	-"-	8.5	0.28	-"-	9.0	0.30	-"-	11.5	0.38
$36 = 6 \times 6$	-"-	5.4	0.15	-"-	10.3	0.29	-"-	11.0	0.30	-"-	14.1	0.39
$49 = 7 \times 7$	-"-	7.6	0.16	-"-	14.5	0.30	-"-	15.8	0.32	-"-	19.8	0.40
$64 = 8 \times 8$	-"-	10.2	0.16	-"-	19.7	0.31	-"-	21.0	0.33	-"-	26.7	0.42

Table 9: Timing for ADI computation with 10 iterations; $N_\epsilon = 16$

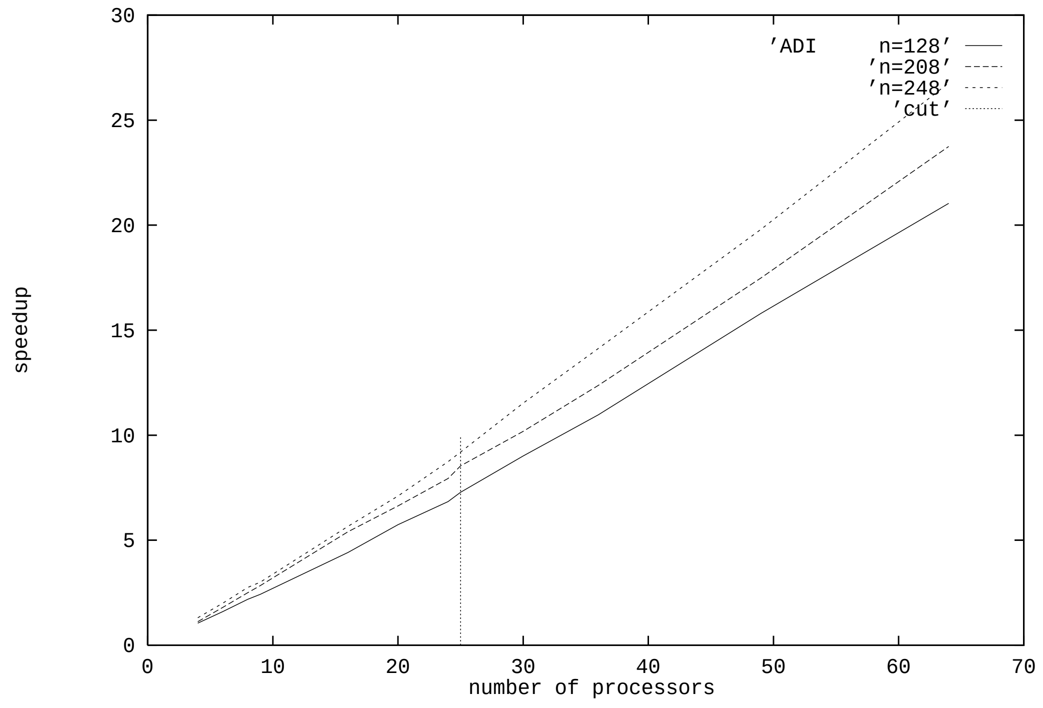

Figure 11: Achieved speedup when the domain decomposed into cells using ADI.

Table 9 presents the computational time and the achieved speedup from using *cell* type computation. When the computation is performed using the *cell* strategy the size of the storage in each processor dictates the size of the experiment. Only $n = 32, 96, 128$ and 224 can fit into the machine. Figure 11 presents the achieved linear speedup for $P \leq 25$. For $P > 25$, the S_p is calculated as before. The upper line ($n = 248$ in Figure 11) is plotted by using the extrapolated values for S_p from different n at each particular P.

The efficiency for the cell computation, as presented in table 9, is not high and achieves only about $15-42\%$. However, the trend is such that E_p grows (almost *linearly*) with the number of points n inside each processor, and also with the number of processors (Figure 12).

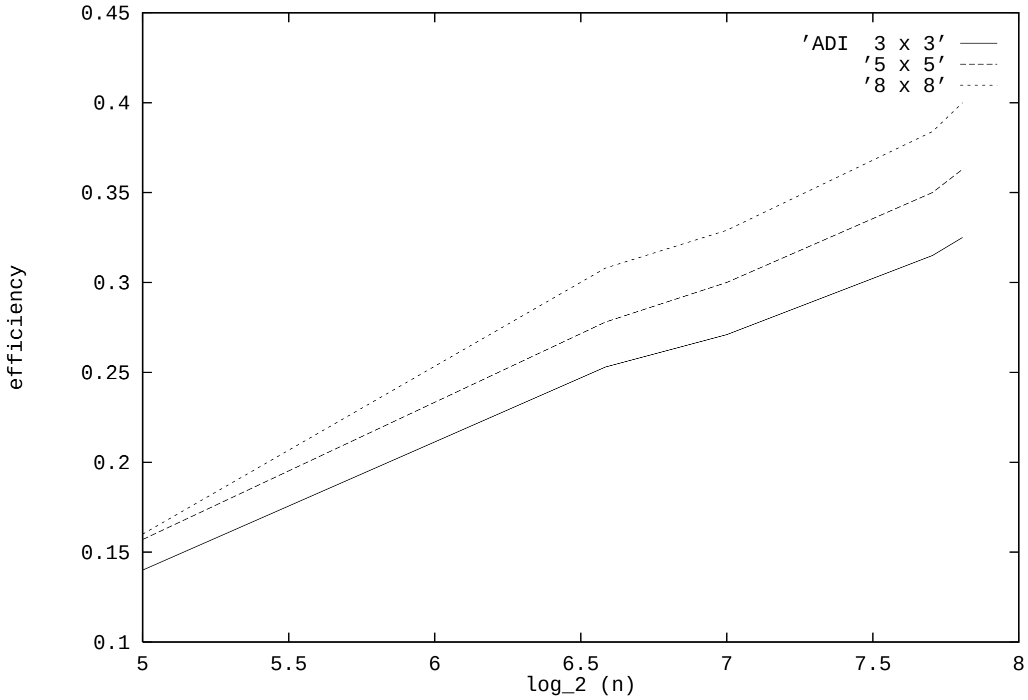

Figure 12: Efficiency when the domain decomposed into cells using ADI.

10 Numerical Accuracy

The accuracy of the current multidomain spectral method with the Local Fourier Basis was tested on several examples. The most simple one is the scalar heat equation

$$\frac{\partial U}{\partial t} = \nabla^2 U + F(x, y) \qquad \text{in } \Omega = (0, 2\pi)^2$$

$$U(x + 2\pi, y) = U(x, y + 2\pi) = U(x, y)$$

$$U(x, y, 0) = 0$$

with the sinusoidal forcing term, corresponding to the exact solution $U_{ex} = \cos k_1 x \cos k_2 y (1 - e^{-(k_1^2 + k_2^2)t})$.

Table 10 presents the maximum error $\varepsilon = \max_{0<x,y<2\pi} |U(x, y) - U_{ex}(x, y)|$ at the asymptotic stage $t > 10$ for several values of the time step Δt in all the cases considered above: for the stiffly stable scheme (5.1) with the domain decomposed into strips (case S) or cells (case C) and the ADI scheme (9.3),(9.4) with the domain decomposed into cells (case ADI). The parameters are: $N_x = N_y = 128$, $N_\epsilon = 16, P = 4$ (case S), $P = 4 \times 4$ (cases C and ADI).

The error becomes smaller when the time step Δt decreases since the accuracy of the local approach is determined by the factor $e^{-\lambda l}$, where $\lambda^2 \propto 1/\Delta t + k^2$ (k - the wave number in Eq. (4.4)) and $l = \min(l_x, l_y)$, $\;l_x = 2\pi/P_x,\; l_y = 2\pi/P_y$). For relatively big Δt the error is smaller for

case	$k_1 = k_2$	$\|U - U_{ex}\|_{max}$		
		$\tau = 0.1$	$\tau = 0.05$	$\tau = 0.01$
S (strips)	1	4.4 (-5)	1.0 (-6)	2.6 (-10)
	12	1.0 (-9)	9.3 (-9)	1.4 (-9)
C (cells)	1	1.6 (-4)	4.1 (-6)	2.7 (-9)
	12	2.4 (-6)	3.3 (-8)	8.6 (-9)
ADI (cells)	1	2.2 (-4)	5.0 (-6)	1.1 (-9)
	12	9.9 (-6)	1.3 (-8)	3.1 (-9)

Table 10: Maximum error at the asymptotic stage $t > 10$ for several values of the time step

larger $k_i, i = 1, 2$. The reason for that is that k contributes to λ, as was mentioned earlier, and improve the conditions of the local approach. When the time step Δt is small enough, the accuracy in all the cases S, C and ADI is almost the same. We notice that the error indicated in table 10 is mostly of the spatial origin. In the initial stage of the evolution process the temporal error dominates the spatial one.

The next example demonstrates the ability of our multidomain local approach to simulate unsteady solution with rather complex behaviour in time and space:

$$v_x = \frac{\partial \psi}{\partial y}, v_y = -\frac{\partial \psi}{\partial x},$$

$$\psi(x, y) = g(t)e^{(\cos \omega x)(\cos \omega y)}$$

$$g(t) = \frac{1}{40}\left(\frac{1}{10}e^{(sin2t - 3sin2\pi t)} + cos(2\sqrt{2}t)\right) + \frac{0,375}{5}$$

$Re = 100, \omega = 5$. This model was suggested in [7] to investigate the efficiency of the multilevel Non-linear Galerkin method for large-scale numerical computations.

Figures 13 and 14 exhibit the dependence of the numerical error $\varepsilon = |U - U_{ex}|/|U_{ex}|$ on time t in the case S (the norm of the discrete function $f_{i,j} = f(x_i, y_j)$ is defined as follows: $|f| = \sqrt{\sum_{i,j} f_{i,j}^2}$). A $3d$ order time discretization scheme [8] is used with the parameters: $P = 4$, $N_x = 128, N_y = 84$, $N_\epsilon = 16$. The time step is $\Delta t = 5.10^{-3}$ for figure 13 and $\Delta t = 5.10^{-4}$ for figure 14. The reduction of the time step by the order of magnitude results in a 3-order decrease in the error. It means that the main contribution to the error (for the values Δt under consideration) is due to the temporal discretization and thus further reduction in Δt will give better accurate result.

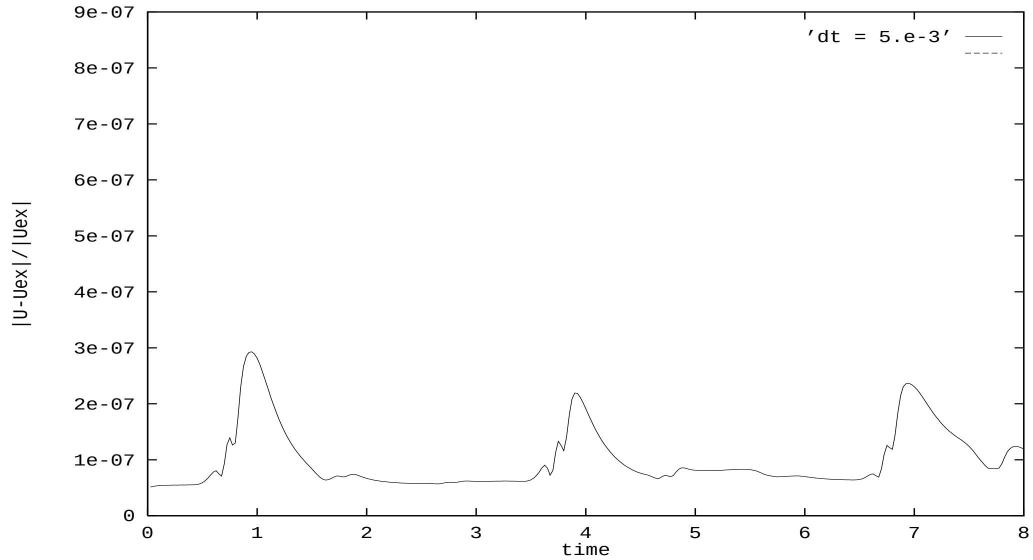

Figure 13: The dependence on the numerical error $\varepsilon = |U - U_{ex}|/|U_{ex}|$ on time t in case of strip computation

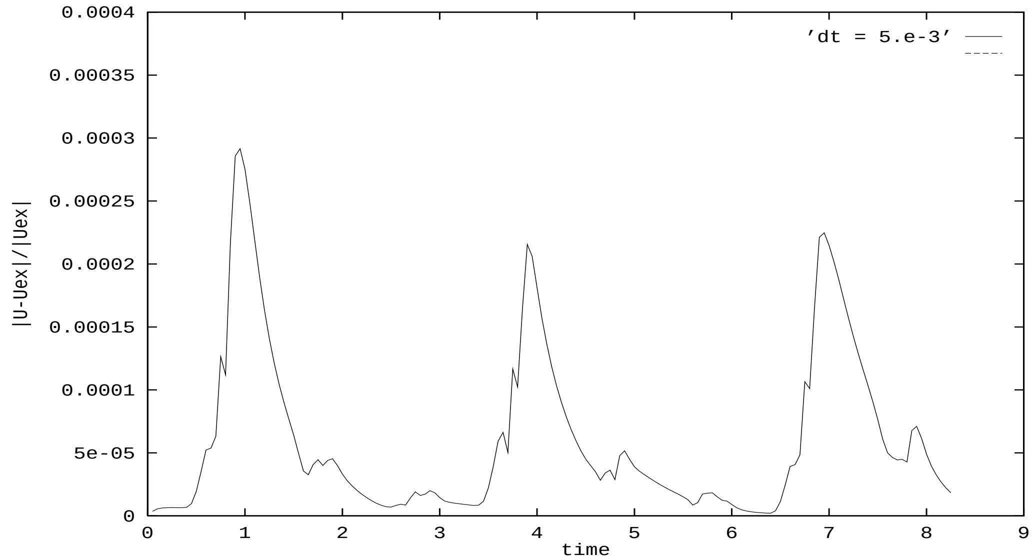

Figure 14: The dependence on the numerical error $\varepsilon = |U - U_{ex}|/|U_{ex}|$ on time t in case of strip computation

Conclusions

We present implementation of parallel algorithms for the numerical solution of nonlinear time-dependent partial differential equations of parabolic type arising from complex large scale problems. The parallelization is achieved by using domain decomposition (DD) techniques. The essential feature of this algorithm is that the spatial discretization in each subdomain is performed by using spectral method with the Local Fourier Basis (LFB). The LFB preserves the spectral accuracy when applied to (nonperiodic) elemental functions in subdomains. Our solutions are based on a special projection technique that employed to localize functions in a smooth way on the extended subdomain. The projection results in a periodic continuous function, as smooth as the original one, after which a spectral Fourier method is applied on a local period to compute the elemental solutions concurrently in all subdomains.

The application of the Parabolic Domain Decomposition (PDD) approach along with the LFB was shown to be very efficient when applied to a 2-dimensional domain splitted into strips and rectangular cells. In that case, *all* matching relations become completely uncoupled (at the price of some overlapping of subdomains required by the LFB implementation). Thus, all communication is reduced to interactions between neighbouring elements and thus fits to scalable message-passing multiprocessor.

2-D Navier-Stokes type modeling equation is implemented on the Meiko message-passing type scalable MIMD multiprocessor.

Detailed performance analysis is presented. The algorithm is scalable in the sense that problems of equal size have the same speedup when the number of processors increase because the communication is reduced from global to local and thus the solution depends on direct neighboring processors. Speedup grows linearly with the number of processors.

The same results are derived when the the alternate direction implicit (ADI) method is used as an efficient time-discretization scheme.

The accuracy of the present multidomain spectral method with the Local Fourier Basis was tested on several examples.

References

[1] A. Averbuch, M. Israeli, L. Vozovoi, Spectral Multidomain Technique with Local Fourier Basis, *J. of Scientific Computing*, Vol. 8, No. 2, pp. 135-149, June 1993.

[2] R. Coifman, Y. Meyer, Remarques sur l'analyse de Fourier à fenêtre, série, *C.R.Acad.Sci.* Paris **312** (1991), p. 259-261.

[3] A. Averbuch, M. Israeli, K. Rubinsky, L. Vozovoi, Implementation of Multidomain Fourier Algorithms for Parallel Solution of the Navier-Stokes Equations, submitted.

[4] M. Israeli, L. Vozovoi, A. Averbuch, Parallelizing Implicit Algorithms for Evolution Equations by Parabolic Domain Decomposition, *J. of Scientific Computing*, Vol. 8, No. 2, pp. 151-166, June 1993

[5] M. Israeli, L. Vozovoi, A. Averbuch, Domain decomposition methods for solving parabolic PDEs on multiprocessors, *Applied Numerical Mathematics*, **12** (1993), pp. 193-212.

[6] F. Jaubertau, C. Rosier, R.Temam, A Nonlinear Galerkin Method for the Navier-Stokes equations, *Computer Method in Applied Mechanics and Engineering*, **80**, 1990, p. 245-260.

[7] F.Jaubertau, C.Roisier, R.Temam, The Nonlinear Galerkin method in computational fluid dynamics, *Applied Numer. Mathem*, **6**, (1989/90), p. 361-370.

[8] G.E. Karniadakis, M. Israeli, S. Orszag, High-order Splitting Methods for the Incompressible Navier-Stokes Equations, *J.Computational Physics* **97**, No. 2, (1991), p. 414-443.

[9] Special Issue: High-Performance Computing and Physics, *Physics Today*, **Vol. 46**, No. 3, March 1993.

[10] R.Temam, Stability Analysis of the Nonlinear Galerkin Method, Math. of Computation **57**, No. 196, (1991), p. 477-505.

[11] S. Orszag, M. Israeli, M. Deville, Boundary Conditions for Incompressible Flows, *J. Scientific Computing*, **1**, (1986), p. 75-111.

[12] L. Vozovoi, M. Israeli, A. Averbuch, Spectral Multidomain Technique with Local Fourier Basis II: Decomposition into Cells, *J. of Scientific Computing*, Vol. 9, No. 3, pp. 311-326, 1994.

[13] L. Vozovoi, M. Israeli, A. Averbuch, Domain Decomposition methods with local Fourier basis for parabolic problems, *The Contemporary Mathematics volume on Domain Decomposition*, edited by Quarteroni et al., Vol. 157, pp.223-230, 1994.

[14] L. Vozovoi, A. Averbuch, M. Israeli, Adaptive Spectral Multidomain Approach to Stiff Problems, to appear.