Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 163–186
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Parallel simulation of axon growth in the nervous system ☆

Jörg Wensch [a], Ben Sommeijer [b,*]

[a] *Fachbereich Mathematik und Informatik, Institut für Numerische Mathematik, Martin-Luther-Universität Halle-Wittenberg, 06099 Halle, Germany*
[b] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

Received 5 June 2002; received in revised form 27 September 2002; accepted 15 April 2003

Abstract

In this paper we discuss a model from neurobiology, which describes the outgrowth of axons from neurons in the nervous system. The model combines ordinary differential equations, defining the movement of the axons, with parabolic partial differential equations. The parabolic equations model the concentrations of chemicals. The axons are guided by the gradients of these chemoattractant and chemorepellant concentrations. We briefly discuss the numerical techniques that we have used to solve this coupled parabolic-gradient system. Special attention is given to the parallel implementation on the SGI Origin 3000 (Teras), a multiprocessor machine. For that purpose we use the OpenMP standard. Several parallelization strategies are introduced and tested on the basis of a test example. Simulation results as well as performance results are reported.

Keywords: Parabolic equations; Gradient equations; Computational neuroscience; Parallel computing

1. Introduction

During the development of the nervous system of a human individual, connections for innervation are formed when axons reach their targets. These axons

☆ The investigations reported in this paper were supported by the NCF (Foundation Nationale Computer Faciliteiten).
* Corresponding author.
E-mail addresses: wensch@informatik.uni-halle.de (J. Wensch), b.p.sommeijer@cwi.nl (B. Sommeijer).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2003.04.006

are sent out by neurons. One of the mechanisms to guide the axons is the diffusion of a chemoattractant secreted by the target. The growth cones of the axons can sense the chemoattractant and grow 'uphill' the concentration gradient. Growing axons often form bundles, which is most likely caused by chemoattractants that are produced by the axons themselves. Upon reaching the target zone, the axons must steer away from each other to innervate their specific target. This debundling is caused by diffusible chemorepellants, also secreted by the axons.

In [3], Hentschel and Van Ooyen describe a mathematical model to study the above processes. In the present paper we will use a slightly modified version of this model. It is based on a set of partial differential equations (PDEs) of parabolic type. Each PDE describes the concentration distribution in space and time of one specific chemical species. Coupled with these parabolic equations we have gradient equations, describing the positions of the axons. These gradient equations are ordinary differential equations (ODEs).

The model will be solved numerically, using techniques similar to those used in [8] (see also [4]). For the spatial discretization of the parabolic equations we use standard second-order finite differences. The gradients are approximated by bilinear interpolation. The resulting semi-discrete system will be integrated in time by a second-order explicit method of Runge–Kutta–Chebyshev (RKC) type [7]. This method has been designed for integrating mildly stiff ODEs with a Jacobian matrix which is 'close to normal' and possesses a spectrum which is located in a narrow strip along the negative real axis in the complex plane. The neuronal problem under consideration fulfils these requirements and the experiments reported in [8] indicate that RKC is an efficient time integration technique.

Based on these numerical ingredients, we have implemented the resulting code on a parallel computer, viz. the SGI Origin 3000, briefly the 'Teras'. The main purpose of the present paper is to report on our experiences with the Teras for this particular problem.

In Section 4 we discuss the various parallelization strategies that we have tested, followed – in Sections 5 and 6 – by the simulation results and the parallel performance of the code on the Teras. These sections are preceded by a brief description of the problem (Section 2) and a short outline of the numerical methods (Section 3). The paper is concluded by a discussion (Section 7).

2. Problem definition

Here, we briefly describe the mathematical model; more details can be found in [3,8].

The concentrations of the three chemicals will be denoted by ρ_t (the target-secreted attractant), ρ_a (the axon-secreted attractant), and ρ_r (the axon-secreted repellant). These concentrations depend on space and time and satisfy the parabolic PDEs

$$\begin{aligned}
\frac{\partial \rho_{\mathrm{t}}}{\partial t} &= D_{\mathrm{t}}\Delta\rho_{\mathrm{t}} - \kappa_{\mathrm{t}}\rho_{\mathrm{t}} + S_{\mathrm{t}},\\
\frac{\partial \rho_{\mathrm{a}}}{\partial t} &= D_{\mathrm{a}}\Delta\rho_{\mathrm{a}} - \kappa_{\mathrm{a}}\rho_{\mathrm{a}} + S_{\mathrm{a}},\\
\frac{\partial \rho_{\mathrm{r}}}{\partial t} &= D_{\mathrm{r}}\Delta\rho_{\mathrm{r}} - \kappa_{\mathrm{r}}\rho_{\mathrm{r}} + S_{\mathrm{r}}.
\end{aligned} \tag{1}$$

Here, the diffusion coefficients D and the κ's in the linear decay terms are given, positive constants. The last term, S, in each equation represents a source term. We adopt the approach advocated in [3] to model this term by a Dirac δ-function, which implies that target cells and growth cones are considered as point sources.

The source term S_{t} models the release of the chemoattractant ρ_{t} in *all* target points. Hence, we choose S_{t} of the form

$$S_{\mathrm{t}} = \sum_{j=1}^{N_{\mathrm{t}}} \sigma_{\mathrm{t}}\delta(\mathbf{x} - \mathbf{x}_j^{\mathrm{T}}), \tag{2}$$

where N_{t} denotes the number of targets, which are located in the fixed spatial points $\mathbf{x}_j^{\mathrm{T}}$. In principle, the factor σ_{t} may depend on all three chemical species. However, since the dependence is not known in detail, σ_{t} will be chosen constant.

For the source terms S_{a} and S_{r} we use similar expressions:

$$S_{\mathrm{a}} = \sum_{j=1}^{N_{\mathrm{a}}} \sigma_{\mathrm{a}}\delta(\mathbf{x} - \mathbf{x}_j^{\mathrm{A}}), \tag{3}$$

$$S_{\mathrm{r}} = \sum_{j=1}^{N_{\mathrm{a}}} \sigma_{\mathrm{r}}\delta(\mathbf{x} - \mathbf{x}_j^{\mathrm{A}}). \tag{4}$$

Here, the summation index runs to N_{a}, the number of axons, and $\mathbf{x}_j^{\mathrm{A}}$ denotes the spatial position of the jth axon. Notice that, due to the movement of the axons, the points $\mathbf{x}_j^{\mathrm{A}}$ change in time, whereas the $\mathbf{x}_j^{\mathrm{T}}$ in (2) are fixed in time. Again, by lack of information, σ_{a} will be chosen constant. For σ_{r} it is reasonable to assume a dependence on ρ_{t}, because secretion of chemorepellants only starts when the axons are in the vicinity of the targets, i.e., when ρ_{t} is large. The precise form of $\sigma_{\mathrm{r}}(\rho_{\mathrm{t}}(\mathbf{x}))$, as well as the values of all other parameters will be specified in Section 5.

The positions $\mathbf{x}_j^{\mathrm{A}}$ of the axons migrating towards the targets are mainly determined by the gradient of the chemoattractant ρ_{t}. In addition to that, the path the axons will follow is also determined by the gradient of the mutual attractant ρ_{a} (causing 'bundling') and by the gradient of the mutual repellant ρ_{r} (causing 'debundling'). Hence, for the axon positions we arrive at the gradient equations

$$\frac{\mathrm{d}\mathbf{x}_j^{\mathrm{A}}(t)}{\mathrm{d}t} = \lambda_{\mathrm{t}}\nabla\rho_{\mathrm{t}}(\mathbf{x}_j^{\mathrm{A}}(t),t) + \lambda_{\mathrm{a}}\nabla\rho_{\mathrm{a}}(\mathbf{x}_j^{\mathrm{A}}(t),t) - \lambda_{\mathrm{r}}\nabla\rho_{\mathrm{r}}(\mathbf{x}_j^{\mathrm{A}}(t),t), \quad j = 1,\ldots,N_{\mathrm{a}}, \tag{5}$$

where the λ's are assumed constant and positive. Notice that the gradients have to be evaluated at the position $\mathbf{x}_j^{\mathrm{A}}$. Since the parabolic equations (1) are solved on a

discrete grid in space, the concentrations are only available at the grid points. Thus, in case an axon is somewhere inside a grid cell, the gradients at $\mathbf{x}_j^{\mathrm{A}}$ have to be interpolated using the grid values of the concentrations. This interpolation will be described in Section 3.2.

In all our experiments we have implemented the above problem (1)–(5) in two spatial dimensions. A square domain has been used which is sufficiently large to justify our choice of homogeneous Dirichlet boundary conditions for the chemical species.

3. Numerical methods

This section shortly outlines the numerical techniques that we have selected to solve the problem. This comprises the spatial discretization of the Laplace operator, the interpolation procedure to approximate the gradients, the way the δ-function in the source terms has been implemented, the time integration technique, and a discussion of a phenomenon that we will call 'self-boost'. These issues will be discussed in the next subsections. Details can be found in [4,8].

3.1. Spatial discretization

The problem will be solved on a square domain in $\mathbb{R}^2$. For the spatial discretization we use a uniform grid with mesh size h. The Laplacian in (1) is approximated by the standard second-order difference stencil

$$[\,1 \quad -2 \quad 1\,]/h^2,$$

applied in both spatial directions. On the boundaries we assume homogeneous Dirichlet conditions.

To calculate the gradients in (5), we first approximate $\partial\rho/\partial x$ and $\partial\rho/\partial y$ in the grid points using the second-order difference stencil

$$[\,-1 \quad 0 \quad 1\,]/(2h),$$

applied to the *discrete* grid function ρ_h, which approximates ρ at the grid points. Then, the resulting approximations for these derivatives will be used to calculate $\nabla\rho_h(\mathbf{x}_j^{\mathrm{A}})$ by means of interpolation, which is discussed in the next subsection.

In passing we remark that this difference stencil cannot be applied to grid points on the boundary. However, the computational domain has been chosen sufficiently large as to avoid the situation that axons reach the boundary. Hence, interpolating in cells adjacent to the boundary will never occur.

3.2. Interpolation

In the gradient equations describing the motion of the axon growth cones we need the derivatives of the concentrations of the three chemicals at the location of the growth cones. However, as said in the preceding subsection, we only have available

the grid function ρ_h, representing the concentration values at the fixed grid points. The discrete gradient operator

$$(\nabla_h \rho_h)_{i,j} := \frac{1}{2h} \begin{pmatrix} \rho_{h,i+1,j} - \rho_{h,i-1,j} \\ \rho_{h,i,j+1} - \rho_{h,i,j-1} \end{pmatrix} \tag{6}$$

is applied to the grid function to obtain approximate grid values $\nabla_h \rho_h$ for the gradient. Next, we apply an interpolation operator I_h yielding an approximation to $\nabla_h \rho_h$ in the point $\mathbf{x}_j^{\mathrm{A}}$. Hence, the discrete analogue of (5) reads

$$\frac{\mathrm{d}\mathbf{x}_j^{\mathrm{A}}(t)}{\mathrm{d}t} = \lambda_{\mathrm{t}} \cdot [I_h \nabla_h \rho_{h,\mathrm{t}}](\mathbf{x}_j^{\mathrm{A}}) + \lambda_{\mathrm{a}} \cdot [I_h \nabla_h \rho_{h,\mathrm{a}}](\mathbf{x}_j^{\mathrm{A}}) - \lambda_{\mathrm{r}} \cdot [I_h \nabla_h \rho_{h,\mathrm{r}}](\mathbf{x}_j^{\mathrm{A}}), \quad j = 1, \ldots, N_{\mathrm{a}}. \tag{7}$$

For efficiency reasons we use local interpolation, i.e., only grid points in the vicinity of the point $\mathbf{x}_j^{\mathrm{A}}$ are used as supporting points. In general the set of supporting points will depend on the grid cell in which the point $\mathbf{x}_j^{\mathrm{A}}$ is located. This set changes when the point $\mathbf{x}_j^{\mathrm{A}}$ crosses the border of a cell, which results in reduced smoothness of the right-hand side function of the differential equation. This may have some impact on the efficiency and reliability of the stepsize selection algorithm in the time integrator.

In our implementation we use a bilinear interpolation approach

$$I_h f(x, y) = a + bx + cy + dxy. \tag{8}$$

The coefficients a, b, c, d of the bilinear functions are determined by interpolating in the four corners of the grid cell the axon is located in. The required values of the gradients in the grid points are determined by the symmetric second-order difference formula as described above. Note that this results in a 12-point stencil for the computation of the gradient: the four corners of the grid cell itself and four additional grid points that enter the formulas for $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$, respectively via the discretized ∇-operator.

3.3. Implementation of the δ-function

Next, we discuss the implementation of the δ-function. This function is replaced by an appropriate continuous function which can be evaluated at the grid points. To ensure that the amount of chemical produced is conserved, the integral of the function is required to be equal to one (cf. [1]). We have considered an exponential function and a tensorproduct linear B-spline approach; in both cases, these functions are chosen grid size dependent.

3.3.1. The exponential function

We use

$$\delta_h(\mathbf{x}) = \frac{\gamma}{\pi} \exp(-\gamma \|\mathbf{x}\|^2), \tag{9}$$

where $\|\mathbf{x}\|$ denotes the Euclidean norm of $\mathbf{x}$ and $\gamma = h^{-2}$. This function has unbounded support. To decrease the computational effort we restrict the support to a

subgrid with 8×8 grid points, such that $\mathbf{x}_j^{\mathrm{A}}$ lies in the center grid cell. The values outside the 8×8-subgrid are at most $\exp(-16) \approx 10^{-7}$ of the maximum value of δ_h and therefore negligible.

3.3.2. Linear B-splines

We replace the delta-function by a tensorproduct of linear B-splines centered at zero with support of size $2h$

$$\delta_h(x,y) = B_h(x)B_h(y), \tag{10}$$

where

$$B_h(x) := \begin{cases} \frac{h-|x|}{h^2} & \text{for } |x| \leqslant h \\ 0 & \text{else} \end{cases}. \tag{11}$$

Suppose the axon $\mathbf{x}_k^{\mathrm{A}}$ lies in the grid cell $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ with $\mathbf{x}_k^{\mathrm{A}} = (x_i + \theta_x h, y_j + \theta_y h)$ where $0 \leqslant \theta_x, \theta_y < 1$. Then the contributions of the approximated δ-function at the grid points are simply the values of $\delta_h(x - x_k^{\mathrm{A}}, y - y_k^{\mathrm{A}})$ evaluated at the four corners of the cell:

$$\begin{aligned} \delta_{h,i,j} &= \frac{(1-\theta_x)(1-\theta_y)}{h^2}, \\ \delta_{h,i+1,j} &= \frac{\theta_x(1-\theta_y)}{h^2}, \\ \delta_{h,i,j+1} &= \frac{(1-\theta_x)\theta_y}{h^2}, \\ \delta_{h,i+1,j+1} &= \frac{\theta_x\theta_y}{h^2}. \end{aligned} \tag{12}$$

This approach also allows for a geometrical interpretation: when we consider the division of the grid cell into four rectangles by the lines $x = x_k^{\mathrm{A}}$ and $y = y_k^{\mathrm{A}}$, then the contribution at a corner of the cell is proportional to the area of the rectangle laying opposite. In fact, this is the approach suggested by Dallon (cf. [1]).

3.4. Time integration

Once the techniques described in the preceding subsections have been applied, we end up with an initial-value problem for a system of ordinary differential equations (ODEs), which will be written in the form

$$\frac{\mathrm{d}U}{\mathrm{d}t} = F(U), \quad t > 0, \;\; U(0) = U_0. \tag{13}$$

For the time integration of (13) we will apply the explicit Runge–Kutta–Chebyshev (RKC) method. This method is especially efficient in case the Jacobian matrix $F'(U)$ is 'close to normal' with eigenvalues in a long, narrow strip along the negative axis in the complex plane. The underlying neuronal problem indeed satisfies these requirements.

RKC is based on the s-stage formula

$$\begin{aligned} Y_0 &= U_n, \\ Y_1 &= Y_0 + \tilde{\mu}_1 \tau F_0, \\ Y_j &= (1-\mu_j-\nu_j)Y_0 + \mu_j Y_{j-1} + \nu_j Y_{j-2} + \tilde{\mu}_j \tau F_{j-1} + \tilde{\gamma}_j \tau F_0, \quad j = 2,\ldots,s, \\ U_{n+1} &= Y_s, \end{aligned} \tag{14}$$

where $F_j = F(Y_j)$. Here, τ denotes the timestep and U_n is an approximation to the exact solution $U(t_n)$, with $t_n = n\tau$. The coefficients are defined as follows. Let T_j denote the Chebyshev polynomial of the first kind of degree j, which satisfies the three-term recursion $T_0(x) = 1$, $T_1(x) = x$, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, $2 \leqslant k \leqslant j$. Defining

$$\epsilon = 2/13, \qquad w_0 = 1 + \epsilon/s^2, \qquad w_1 = \frac{T_s'(w_0)}{T_s''(w_0)},$$

$$b_j = \frac{T_j''(w_0)}{(T_j'(w_0))^2} \quad (2 \leqslant j \leqslant s), \quad b_0 = b_2, \quad b_1 = b_2,$$

the coefficients in (14) are given by

$$\tilde{\mu}_1 = b_1 w_1, \qquad \mu_j = \frac{2b_j w_0}{b_{j-1}}, \qquad \nu_j = \frac{-b_j}{b_{j-2}}, \qquad \tilde{\mu}_j = \frac{2b_j w_1}{b_{j-1}},$$

$$\tilde{\gamma}_j = -(1 - b_{j-1}T_{j-1}(w_0))\tilde{\mu}_j \quad (2 \leqslant j \leqslant s).$$

Notice that these coefficients are available in analytical form for arbitrary $s \geqslant 2$. The real stability boundary $\beta(s)$ is *quadratic* in s and is given by $0.65s^2$. We remark that s can vary, which implies that (14) is in fact a whole family of integration formulas. A nice property of RKC is that the local error coefficients are almost independent of s. Hence, based on error control, RKC determines first the largest possible timestep τ with respect to accuracy requirements. Then, the most efficient stable formula (i.e., the one with minimal s-value) is selected to satisfy the linear stability condition. Another important property of the family is due to the three-term recursion for Chebyshev polynomials. This makes it possible to implement the integration formula with only a few vectors for storage, no matter the number of stages.

Finally, we remark that the approximation U_{n+1} as well as the intermediate approximations Y_j $(j = 2,\ldots,s)$ are second-order accurate. Source code for RKC can be obtained from the address ftp://ftp.cwi.nl/pub/bsom/rkc.

3.5. Avoiding self-boost

We have applied the RKC time integration scheme to the semi-discretized problem with several interpolation formulas for the gradient and several approaches for the source terms. For some combinations we encountered unsatisfactory results: the system approaches a stationary point where single axons stop near their starting points far away from the targets while other axons reach the targets.

To obtain a better understanding of this phenomenon we consider a single axon moving under the influence of a gradient of only one chemical ρ (see also [4] for a more comprehensive discussion on this topic):

$$\begin{aligned} \dot{\rho} &= L\rho + S(\mathbf{x}^{\mathrm{A}}), \\ \dot{\mathbf{x}}^{\mathrm{A}} &= \nabla|_{\mathbf{x}^{\mathrm{A}}}\rho. \end{aligned} \tag{15}$$

Here, L denotes the diffusion operator. When this equation is subjected to a space discretization then ρ, L, S, and ∇ are replaced by their discrete analogues ρ_h, L_h, S_h, and ∇_h, respectively.

Recall that our model assumes that the axon-secreted chemicals control the bundling and debundling of the axons, i.e., axons are influenced by the chemicals secreted by other axons. There is a side-effect we have not yet taken into account: the chemicals that are secreted by a particular axon induce a movement of that axon itself. We denote this undesirable behaviour by the term 'self-boost'.

In order to avoid the 'self-boost' the discretized version has to satisfy

$$\nabla_h|_{\mathbf{x}^{\mathrm{A}}} S_h(\mathbf{x}^{\mathrm{A}}) = 0. \tag{16}$$

In this condition, three processes are involved: distributing the source term in $\mathbf{x}^{\mathrm{A}}$ over the neighbouring grid points, the approximation of the first derivatives of the contributions in these grid points, and the interpolation in the point $\mathbf{x}^{\mathrm{A}}$ of these derivatives. In fact, the compatibility condition (16) requires that the overall effect of the *combination* of these processes vanishes, thus avoiding self-boost of the axon.

We will now show that the discrete gradient operator (6), in combination with the bilinear interpolation (cf. (8)) and the implementation of the δ-function as described in Section 3.3.2 indeed satisfy the condition (16). We restrict ourselves to the gradient in x-direction (the computation in y-direction is completely analogous): using the interpolation procedure, ∂_x in the point $\mathbf{x}^{\mathrm{A}}$ can be expressed in terms of ∂_x in the four surrounding grid points:

$$\begin{aligned} \partial_x|_{\mathbf{x}^{\mathrm{A}}} S_h(\mathbf{x}^{\mathrm{A}}) = \{&(1-\theta_x)[(1-\theta_y)\partial_{x,i,j} + \theta_y \partial_{x,i,j+1}] \\ &+ \theta_x[(1-\theta_y)\partial_{x,i+1,j} + \theta_y \partial_{x,i+1,j+1}]\} S_h(\mathbf{x}^{\mathrm{A}}). \end{aligned}$$

Then, replacing ∂_x by the discrete analogue (6) yields for the right-hand side

$$\frac{1}{2h}\{(1-\theta_x)[(1-\theta_y)\delta_{h,i+1,j} + \theta_y \delta_{h,i+1,j+1}] - \theta_x[(1-\theta_y)\delta_{h,i,j} + \theta_y \delta_{h,i,j+1}]\}.$$

Finally, we substitute the δ_h-values in the grid points as defined in (12) to obtain

$$\begin{aligned} &\frac{1}{2h}\frac{1}{h^2}\{(1-\theta_x)[(1-\theta_y)\theta_x(1-\theta_y) + \theta_y\theta_x\theta_y] \\ &\quad - \theta_x[(1-\theta_y)(1-\theta_x)(1-\theta_y) + \theta_y(1-\theta_x)\theta_y]\}, \end{aligned}$$

which is readily verified to vanish indeed.

4. Parallelization strategies

When the space grid is refined in both x- and y-direction the computational effort rises approximately as $\mathcal{O}(h^{-3})$. Two powers of h^{-1} are caused by the larger number of grid points while a further factor h^{-1} originates from the increased stiffness of the system that forces RKC to use more stages to keep the integration process stable (using the same time step). Needless to say that the use of a parallel computer is indispensable for very fine meshes.

Having prescribed the spatial discretization and the time integration, essentially two components of the total algorithm are suitable for parallelization: the computation of the right-hand side function and the vector operations in RKC.

4.1. The Teras architecture

The Teras machine has 1024 processors, each of which has a theoretical peak performance of 1 Gflop/s. They are organized in two clusters of 512 processors where each cluster is organized as a hypercube. Small units of four processors share physically the same memory via a so-called HUB. Larger units are built from smaller ones on the basis of a hypercube-structure, where the communication is organized by routers. Therefore blocks with more than 4 processors have *logically* shared memory, but *physically* distributed memory. This fact has to be taken into account when building a parallel application. More information about the Teras machine can be found in [6].

4.2. Parameterisation

Before discussing the actual parallel implementation, we first give a parameterisation for this particular application and for the parallel machine (Teras) that we have used. This is a useful exercise since it provides information to predict the behaviour of a future 3D implementation. Furthermore, such a parameterisation can be of use in tailoring computer configurations to the specific features of the application.

Following the approach described in [2], we will discuss two important parameters: γ_a, characterizing the application and γ_m, characterizing the parallel machine. The smaller the quotient γ_m/γ_a, the better the application fits the parallel configuration. By that we mean that the communication requires a small part of the overall time.

The parameter γ_a is defined as the number of operations (in Mflops) that has to be performed on a single processor divided by the amount of data (in Mwords) that has to be transferred from one processor to the other processors. Obviously, this number depends on the application as well as on the algorithm. Based on the space and time discretization methods discussed in Section 3, this parameter can, in first approximation, be quantified as

$$\gamma_a = \frac{25.5N}{p}, \tag{17}$$

where N denotes the number of grid points in each of the two spatial dimensions and p is the number of processors. In this expression we only took into account the work and transfer corresponding to the $3N^2$ grid point values. Since the contribution of the N_a axons and N_t targets is two orders of magnitude smaller, it has been omitted in (17).

The parameter γ_m essentially describes 'how many operations can be performed on a processor during the time needed to send one word of data from one processor to another'. More details, as well as a precise definition of γ_m, can be found in [2]. Since de Teras machine is of so-called CC-NUMA (Cache-Coherent Non Uniform Memory Access) architecture, the parameter γ_m is not a constant. This is due to the fact that the communication time depends on the 'distance' between processors (i.e., the number of routers that have to be passed). As a consequence, running in a multi-users situation (as we did) may result in different γ_m-values. Basically, the γ_m-value can be computed by taking the quotient of the effective processor performance and the network bandwidth. For the Teras system that we used, we then arrive at the value 20. Due to latency and 'variable distance' effects, $\gamma_m = 25$ seems to be a realistic choice. As we shall see in Section 6, the effect of a non-constant γ_m is clearly noticeable in case of a 'subdomain'-approach (this concept will be discussed in Section 4.4.1) when running on the coarsest spatial mesh using 13 processors. Only for this case, repeated runs showed that the time needed for communication was quite unpredictable. However, in all other cases (i.e., different approaches to distribute the grid points, or finer meshes and more processors) revealed that the run times show only modest variation, particularly if the number of processors was chosen as a power of 2.

4.3. The shared memory model on the Teras

We have decided to use the OpenMP standard [5] to implement our application. The statements influencing parallelization are hidden as compiler directives in comment lines so that the code runs on a single-processor machine, too. These statements include the specification of parallel regions, synchronization statements and data type specifications. When execution reaches the beginning of a parallel region then on each processor a separate thread is created. The tasks inside the region are distributed among the threads by three different models:

- PARALLEL DO for the parallelization of loops. Load-balancing is supported by different scheduling modes:
  - STATIC: suitable for loops with constant cost per cycle; processors start simultaneously.
  - DYNAMIC: suitable for loops with nonconstant costs per cycle.
  - GUIDED: suitable for loops with constant costs per cycle; processors start at different times.
- PARALLEL SECTIONS for multiple code parallelization, i.e., different code runs in parallel on different processors.
- PARALLEL for single code parallelization, i.e., the same code runs on multiple processors. In this mode each processor needs private initialization data. These

are supported by the THREADPRIVATE clause, which allows private copies of a common-block.

The distribution model may change inside a parallel region or even inside a subroutine (so-called 'orphane parallelism'). In parallelizing the right-hand side we have mainly utilized the parallelization of loops. The parallel version of RKC works with the single-code approach and orphane parallelism.

In contrast to MPI no explicit communication is required. Data communication is made obsolete by SHARED variables, which each processor is allowed to read and to modify. The (principal) other type of variable is PRIVATE, for which each processor maintains a separate copy. The parallel computation of sums (or similar operations) is supported by a REDUCTION clause. Synchronization among the processors is supported mostly implicitly, but can also be forced explicitly by setting a BARRIER or marking regions as CRITICAL.

The strength of OpenMP is that a running serial code can be converted into a running parallel code by incremental steps, keeping the code running in the intermediate stages.

4.4. The right-hand side function

The right-hand side function consists of three main parts:

- the discrete Laplacian
- the interpolation of the gradients
- the discretized δ-function

4.4.1. Parallelizing the Laplacian

We have to compute second derivatives of the three concentrations ρ_j, $j = 1, 2, 3$, in each grid point $(x_k, y_l), k,\ l = 1, \ldots, N$ (for notational convenience, ρ_t, ρ_a and ρ_r are denoted by ρ_j, $j = 1, 2, 3$ in this subsection).

The natural approach is to assign to each processor a subset of the triples (j, k, l) where all subsets have approximately the same size. Using p processors we simply design a loop that runs over all s triples (j, k, l) and schedule the cycles $0, \ldots, \lfloor s/p \rfloor - 1$ on processor number 0, the cycles $\lfloor s/p \rfloor, \ldots, \lfloor 2s/p \rfloor - 1$ on processor 1, and so on. We have simply to decide on the design of the loop.

We have selected three different approaches that are illustrated in Fig. 1. Naturally, the diffusion term corresponding to a single grid point is computed on a single processor. The only decision we have to make is how to distribute the grid points among the different CPUs. The element-wise approach is equivalent to OpenMP-style static scheduling of the contiguous vector of the grid point values for all three chemicals. The column-wise approach is almost identical but differs in two minor points. First, it schedules complete columns. Second, the master processor (number zero), which also hosts the axon locations, takes care of the last set of grid points.

The third approach uses a domain-splitting style. The computational domain is partitioned into M^2 square subdomains. To each of these squares and each chemical

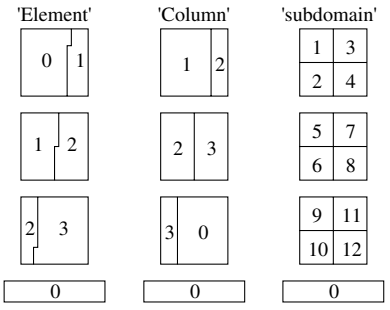

Fig. 1. Different parallelization strategies for the grid points.

we assign a separate CPU. Finally the master processor is assigned to the axon coordinates. This results in a total of $3M^2 + 1$ processors.

4.4.2. Parallelizing the interpolation of the gradients

This task is parallel distributed over the axons, i.e., each processor works on a subset (which may be empty in case $p > N_a$) of the set of axons. Note that here as well as in the computation of the Laplacian the processors write on completely disjunct subsets of the solution vector. For all shared data the access is read-only. Under this condition the parallelization of loops is straightforward. The opposite case is called a race-condition: multiple processors update one memory location, which can lead to different results, depending on the order in which the processors access the variable.

4.4.3. Parallelizing the source terms

The computation of the source terms offers a considerable amount of inherent parallelism; however, it is rather difficult to fully exploit this parallelism. The most natural approach is to schedule the axons and targets in parallel. There is no problem with parallel scheduling of the targets because their location is fixed. However, the axons may be located very close together or even in the same grid cell, which implies that different axons give contributions to the concentrations in the same grid points. We get race-conditions, resulting in additional synchronization. OpenMP offers a temptingly simple way to synchronize—the ATOMIC and CRITICAL clauses. Whereas the first protects one statement from race-conditions, the latter one protects a region of code.

When we use the exponential approach for the source terms, the computational effort is quite substantial because the computation of exponentials is costly compared with additions and multiplications. Using 40 axons and an 8×8-subgrid as support we have 2560 grid points to update. An ATOMIC clause guarantees that only one grid point at a time is updated. The processors set so-called locks to check whether the protected clause can be safely executed. In this way, however, we burden additional communication to the system. To our experience, this approach does not

give any speedup compared with sequential computation. The synchronization by an ATOMIC clause or a CRITICAL clause should be used with great care.

We briefly discuss a few more sophisticated approaches that we did not work out in detail. The first approach exploits parallelism over the grid points. It utilizes up to 64 processors. We compute the source contributions in the 64 grid points of the support in parallel on up to 64 processors for each axon concurrently. A second, less obvious parallelization approach uses parallelism over the axons. It works with up to N_a processors and requires additional synchronization. We assign each processor to an axon. Then at step k on processor l (assigned to axon number l) we compute the contribution at the grid point (i,j) with $8i+j \equiv k+l \bmod 64$. This schedule guarantees that no race-conditions occur as long as the processors are synchronized after each step.

More simple is the parallelization of the tensorproduct B-splines – we simply do not parallelize them at all. The reason is that the computational costs are very low. Our experiments showed that there is no speedup when distributing the tasks on different processors.

4.5. Vector operations in RKC

The code RKC uses five vectors of size n to store internal information when integrating a system of ODEs of dimension n. With these vectors the following $\mathcal{O}(n)$ operations are performed

$$
\begin{aligned}
&u = v,\\
&u = v + \alpha w,\\
&u = \alpha u + \beta v + \gamma_1 w_1 + \gamma_2 w_2 + \gamma_3 w_3,\\
&\alpha = \|v - w\|_u = \max_i \frac{|v_i - w_i|}{a + r * \ |u_i|}.
\end{aligned}
$$

The first three expressions are parallelized by simple loop parallelization. The fourth is parallelized with a REDUCTION clause.

It is important to distribute the work in all loops the same way to avoid data transfer. To implement this we have chosen an approach where each processor uses a THREADPRIVATE common-block to store the information on which part of the vector it operates. This part is fixed for all vector operations in RKC. When a first initialization of a variable is executed by a processor then the machine assigns that variable to the main memory of that processor (this is true even for separate array elements). We initialize each of the five vectors in RKC as well as the vector with the initial values by the same procedure.

Clearly, the optimal load balancing is obtained when each CPU processes the same number of elements of the vectors. But on the other hand we have to find a load balanced parallelization when we compute the right-hand side. Note that the ith component of the argument U and the ith component of the right-hand side $F(U)$ (cf. (13)) are stored on the same memory block. This situation is ideal for the diffusion terms but less favourable for the gradients and the source terms. The

```
MODULE parallel
      common/pdata/iproc,nprocs,ind0,ind1
      !$OMP THREADPRIVATE(/pdata/)
contains
      subroutine initproc()
         use problem
         nprocs=OMP_GET_NUM_THREADS()
         iproc=OMP_GET_THREAD_NUM()
         call getrange(iproc,nprocs,nvar,ind0,ind1)
          ...
```

Fig. 2. Typical init-routine in module parallel.

gradients depend on the concentrations of the chemicals and occur in the right-hand side of the axon-equations, whereas the source terms depend on the axons and enter the right-hand side of the chemical-equations. In order to avoid additional communication we had to assign the tasks to the processor where the data reside, which could mean that in the case when all axons are very close together one processor has to do all the work.

When using RKC in a parallel environment the user has to parallelize both the right-hand side and the vector operations. The parallelization of the right-hand side is under complete control of the user. To parallelize the vector operations the module 'parallel' has to be modified. In particular, there has to be written an init-procedure and two loops. One loop executes independent operations on a vector in parallel. The other loop computes the norm of a vector in parallel and therefore needs a REDUCTION-clause. For the simple case that each processor works on a contiguous part of the vector we have given these two loops in Fig. 3.

The corresponding init-routine for the element-wise parallelization (simplest case again) is given in Fig. 2. The init procedure has to be called in parallel from the main program. For each individual processor it examines the parallel environment (total

```
subroutine genlinalg(task,y1,y2,a1,a2,a3,a4,a5)
   !$OMP PARALLEL PRIVATE (i)
   do i=ind0,ind1
      call genlinalg_i(task,i,y1,y2,a1,a2,a3,a4,a5)
   end do
   !$OMP END PARALLEL
    ...

subroutine redlinalg(task,err,y1,a1,a2,i1)
   !$OMP PARALLEL REDUCTION(MAX:err) PRIVATE(i)
   do i = ind0,ind1
      call redlinalg_i(task,i,err,y1,a1,a2,i1)
   end do
   !$OMP END PARALLEL
    ...
```

Fig. 3. Sample code from module parallel.

number of processors, number in team) and computes the start- and endpoints ind0, ind1 depending on the individual processor number in a subroutine getrange( ). These individual processor data are stored in the private common-block/pdata/. It can be accessed from each program part that uses the MODULE parallel. In general in this common-block there will be stored additional data that are needed for the parallelization of the right-hand side $F(U)$. Further, when data are distributed in noncontiguous blocks over the processors additional statements in the two routines in Fig. 3 may become necessary. However, all changes are restricted to the user-supplied right-hand side and the MODULE parallel. The parallel version of RKC can be compiled without modifications.

5. Simulation results

In this section we report on the results of the numerical simulations. First, the test example will be defined.

For the computational domain we choose the unit square of size 1 mm^2. This domain has been covered by four different uniform grids of increasing resolution, viz., $N = 64, 128, 256, 512$, where N is the number of grid points in each spatial direction. Since we want to investigate the influence of the grid size on the numerical solution (i.e., convergence aspects) as well as on the parallel performance (see Section 6), we did experiments on all four grids.

The test model is defined by Eqs. (1)–(5), where the parameters have been given the following values:

$$D_t = D_a = D_r = 10^{-4}, \qquad \kappa_t = \kappa_a = \kappa_r = 10^{-4}, \qquad \lambda_t = 10^{-5},$$
$$\lambda_a = 5 \times 10^{-6}, \qquad \lambda_r = 3.75 \times 10^{-5}, \qquad \sigma_t = 6 \times 10^{-5},$$
$$\sigma_a = 7.5 \times 10^{-5}, \qquad \sigma_r = \frac{7.5 \times 10^{-5} \cdot r}{2.42 + r}, \qquad r = 1 - e^{-\rho_t}.$$

Notice that all parameters have a fixed value, except for σ_r which is space- and time-dependent (see also the discussion in Section 2).

In our experiments we let 40 axons ($N_a = 40$) find their way towards 50 targets ($N_t = 50$). We implemented the strategy that a target is 'switched off' as soon as an axon has reached this particular target. This strategy determines the length of the time integration interval: once that all axons have found a free target, the integration stops.

In Fig. 4 we plot the solution obtained on the finest (512×512) grid. The figure shows the typical behaviour of the axons: bundling, moving towards the target region, debundling, and finally connecting to the targets. We mention that the solutions obtained on the coarser grids are *qualitatively* of a similar structure. In this figure, we marked the positions of the axons at three different points in time, viz. at $t = 1200$, 5600, and 7880. At time $t = 1200$, the axon bundle is almost formed. The axons increase the production of the repelling chemical until a balance between the attracting and repelling chemical is obtained. Shortly after time $t = 5600$, the

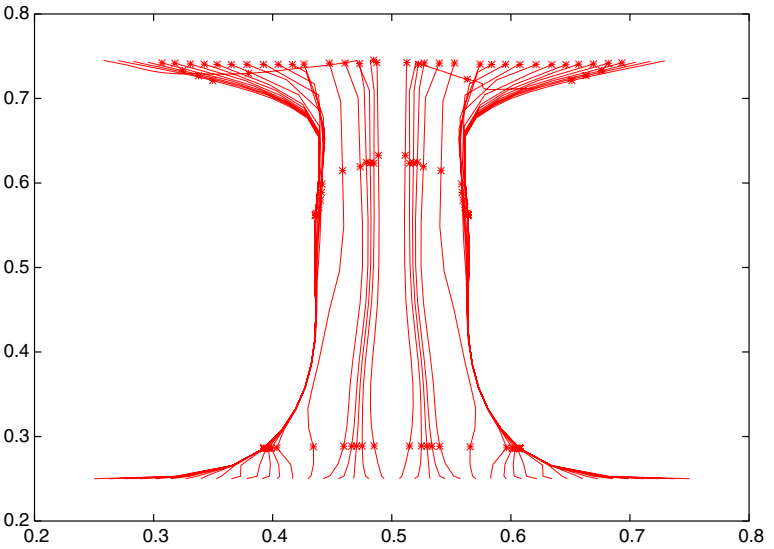

Fig. 4. The path of the axons and their position at $t = 1200$, 5600, 7880 (denoted by x). These simulation results are obtained on the finest (512×512) grid. Unit of length is mm.

repelling chemical dominates the motion and the axons start to debundle. Finally, at $t = 7880$, almost all axons have reached a target. Note that there were more axons running to the center of the target region than the number of free targets. A few axons could not connect within the center region – they find a free target at the outside regions.

In Figs. 5–7 we plot, for these three points in time, respectively, the corresponding concentration fields ρ_t, ρ_a, ρ_r. Additionally, in each figure we show $\rho_{\text{resultant}} := \lambda_t\rho_t + \lambda_a\rho_a - \lambda_r\rho_r$. Notice that the gradient of $\rho_{\text{resultant}}$ defines the right-hand side of the ODEs for the axons. In all three figures we recognize peaks in the concentration $\rho_{\text{resultant}}$ near the axon locations caused by the fact that diffusion is very fast compared with the axon movement. Note that the peaks in the expression $\rho_{\text{resultant}}$ change from upwards in Fig. 5 to downwards in Fig. 6. This is caused by the increased production of repellant. Furthermore, the movement of the axons is mainly controlled by the target secreted attractant, whereas the axon secreted chemicals act as a lower order correction that controls bundling and debundling (which in fact is a movement perpendicular to the main growth direction). At time $t = 7880$ (Fig. 7), when almost all axons have reached a target, we see the concentration of the target secreted attractant dropping down because almost all targets have gone inactive. There are only a few active targets left at the boundaries of the target region.

The solution of the resulting ODE system turns out to be quite sensitive to small perturbations of the parameters and the starting values. This especially holds for the path the axons will follow. Small changes in the positions at the moment that the

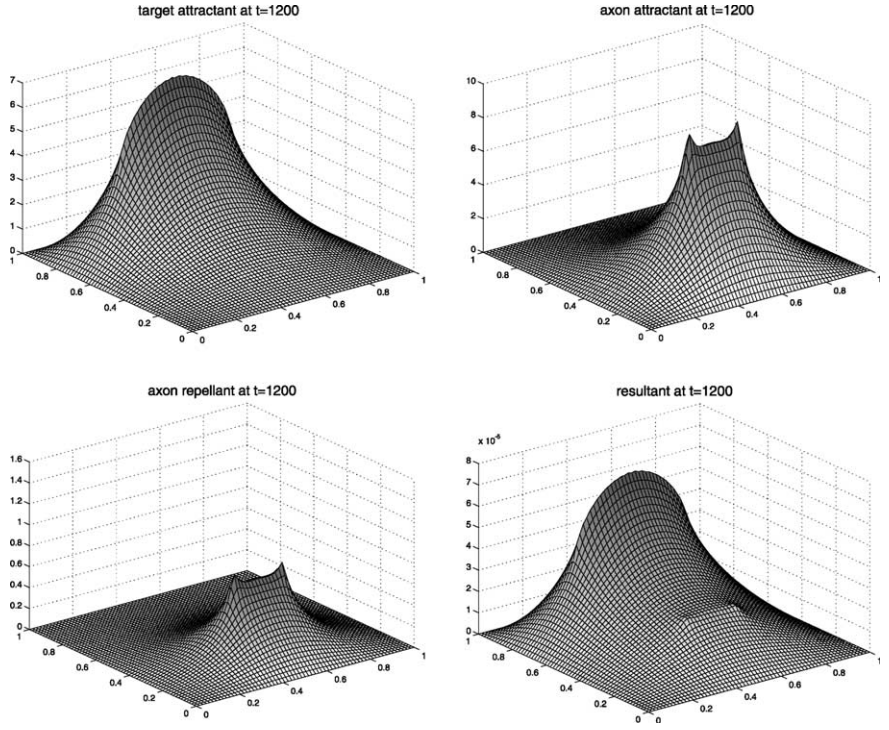

Fig. 5. The dimensionless concentrations ρ_t, ρ_a, ρ_r and the field $\rho_{resultant}$ shortly after the start ($t = 1200$).

axons start debundling may result in substantial differences w.r.t. the moment the axons reach the target region – some axons may even connect to a different target. The same effect can be caused by a change of the resolution of the spatial grid.

Therefore, the solution of this problem requires a very fine mesh, which results in a high dimension of the system of ODEs as well as in increased stiffness. When the grid size is decreased by a factor of 2, then the computational effort increases by a factor of 8, approximately.

6. Parallel performance

We have performed a large number of test runs in which we have varied over the spatial resolution, over the three different approaches 'element', 'column', 'subdomain' (see Fig. 1), and over the number of processors. For each combination we measured the performance by means of CPU-time, speedup factor, and efficiency. Here, we distinguish between overall performance, and the separate performance of the three main parts in the code, viz., the vector operations in RKC, the computation of the diffusion operator, and the interpolation of the gradient. All this

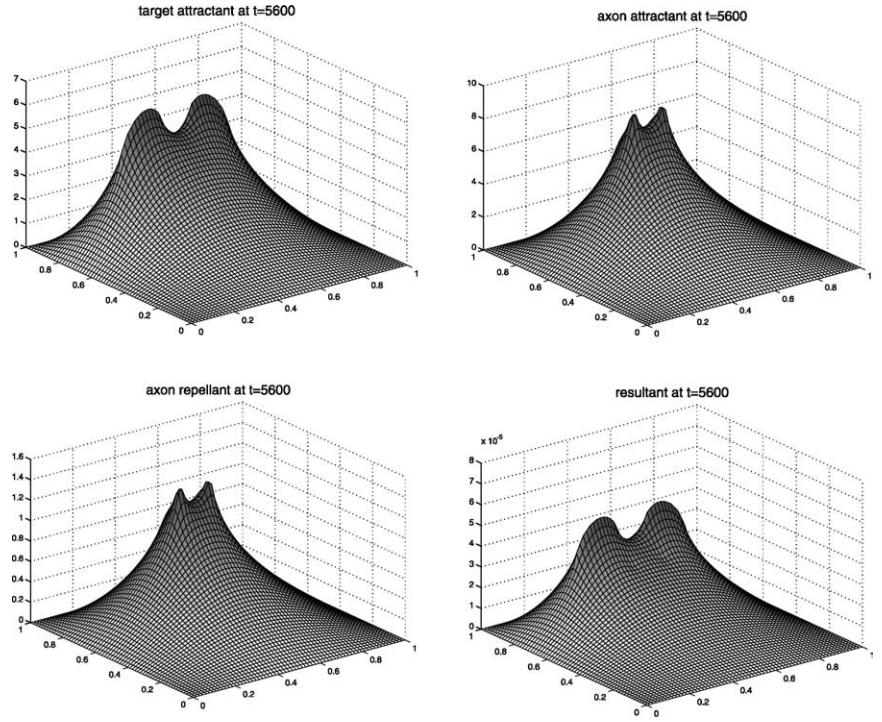

Fig. 6. The dimensionless concentrations ρ_t, ρ_a, ρ_r and the field $\rho_{resultant}$ as the axons are about to debundle ($t = 5600$).

information is collected in Figs. 8–11 (each figure corresponds to a particular spatial grid).

Before stating the conclusions from these experiments, we first consider the characteristic parameters γ_a and γ_m, as discussed in Section 4.2. In [2] it is argued that $\gamma_a/\gamma_m \geqslant 4$ leads to high efficiency, that is, communication takes only a small fraction of the overall time needed. In our application we see that γ_a/γ_m evaluates roughly to N/p. The experiments shown in Figs. 8–11 confirm the assertion made in [2]. Results corresponding to larger values of N/p show even improved efficiency.

Summarizing, the following conclusions can be drawn:

- Using a small number of processors, say up to 8, it is relatively easy to obtain an efficient parallel process: on the coarsest grid, the overall efficiency is $\approx$30%, on the next finer grid $\approx$50–60% efficiency is obtained, whereas the two finest grids even show superlinear speedup factors.
- Concentrating on the finest grid, at least 75% efficiency can be obtained for a number of processors up to 64.
- Comparing the various approaches ('element', 'column', 'subdomain'), we observe that, in many cases, the 'element' and 'column' modes are more efficient than the 'subdomain' mode.

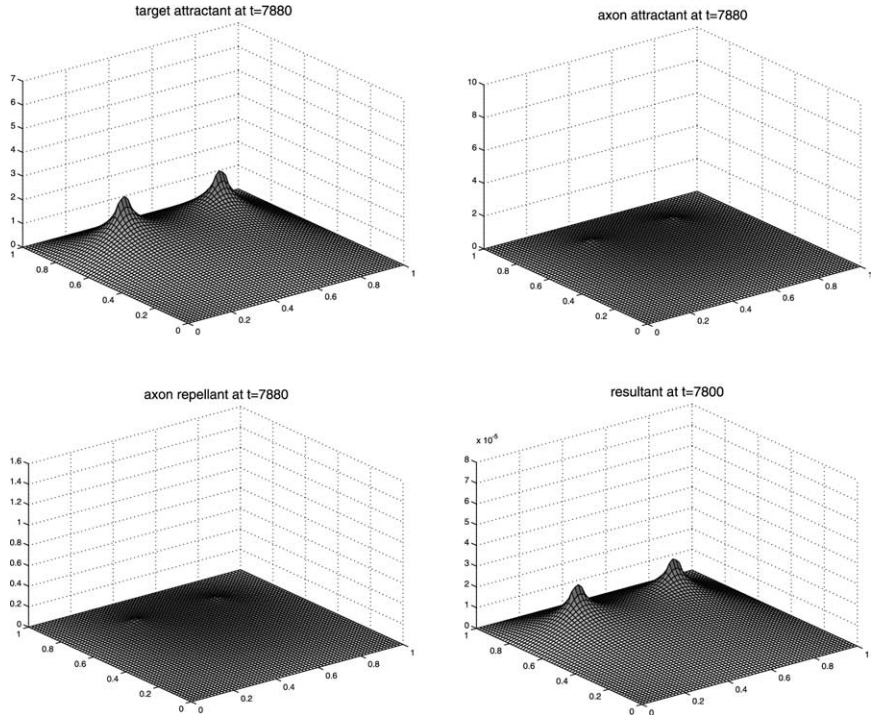

Fig. 7. The dimensionless concentrations ρ_t, ρ_a, ρ_r and the field $\rho_{\text{resultant}}$ close to the end ($t = 7880$).

• On the coarsest grid (64×64), the 'subdomain' approach demonstrates an irregular behaviour: using 13 processors (i.e., $M = 2$, as shown in Fig. 1), a poor performance is obtained. Furthermore, runtimes vary strongly around a factor of 4. A possible explanation is that the system responds to a call for an 'odd' number of processors like 13 with a set of processors that is randomly distributed over the complete hypercube, maybe because 13 processors cannot form a hypercube. The random distribution results in completely unpredictable effort for communication. This is due to the CC-NUMA architecture of the Teras machine, as discussed in Section 4.2. The situation is less drastic on finer grids or for a higher number of processors, like $28 = 3 * 3^2 + 1$.

• When the number of processors is a power of two then the runtimes are much better reproducible. We performed several runs but did not encounter differences of more than 10%.

• With respect to the various subtasks in the code, we conclude that the parallel performance of the vector operations in RKC (the most expensive part of the code) is highly efficient. The parallel computation of the diffusion operator is of reasonable efficiency. The efficiency in the interpolation of the gradients (the least expensive part of the three main tasks) however, strongly decreases as the number of processors increases.

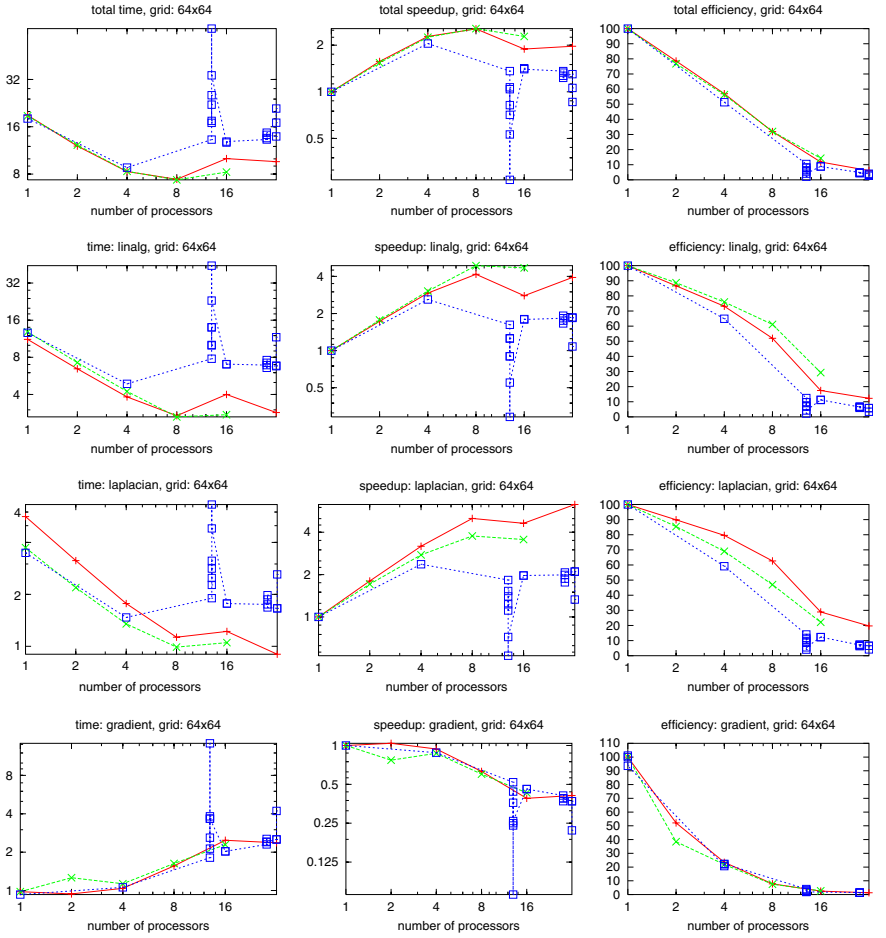

Fig. 8. Results on the 64×64 grid for the 3 approaches (see Fig. 1): 'element' (+++), 'column' ($\times\times\times$), and 'subdomain' ($\square\square\square$). First row: overall results; second row: results for the vector operations in RKC; third row: results for the Laplacian; last row: results for the gradients. In the first column we give the total CPU time in seconds, for various numbers of processors. The second column shows the corresponding speedup factors and the last column the corresponding efficiencies.

7. Discussion

We have discussed the parallel implementation on the SGI Origin 3000 (Teras) of an application originating from neurobiology. The model consists of a set of three parabolic PDEs in two spatial dimensions, coupled with ODEs describing the positions of axons in the spatial domain of the PDEs. After explaining the various numerical techniques that we have used to solve the system, we suggest several parallel approaches to implement the discrete problem. As it turns out, the approach

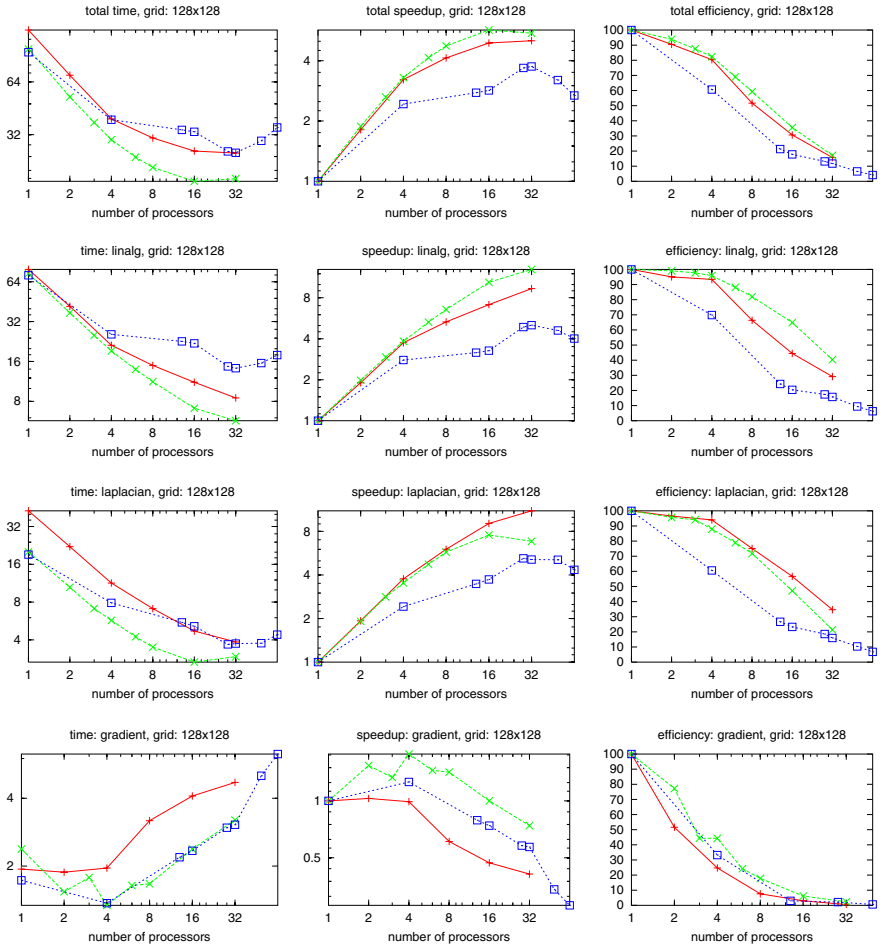

Fig. 9. Results on the 128×128 grid for the three approaches (see Fig. 1): 'element' (+++), 'column' (× × ×), and 'subdomain' (□□□). First row: overall results; second row: results for the vector operations in RKC; third row: results for the Laplacian; last row: results for the gradients. In the first column we give the total CPU time in seconds, for various numbers of processors. The second column shows the corresponding speedup factors and the last column the corresponding efficiencies.

based on subdomains shows irregular behaviour (as a function of the number of processors) and is, in general, less efficient than the approach in which a static scheduling of the contiguous vectors has been used.

The program has been tested on spatial grids of increasing resolution: 64×64, 128×128, 256×256, and 512×512 grid points. As a general conclusion we might say that an efficient parallel implementation can be obtained on the finest grid: >75% overall efficiency for up to 64 processors. On the next coarser grid

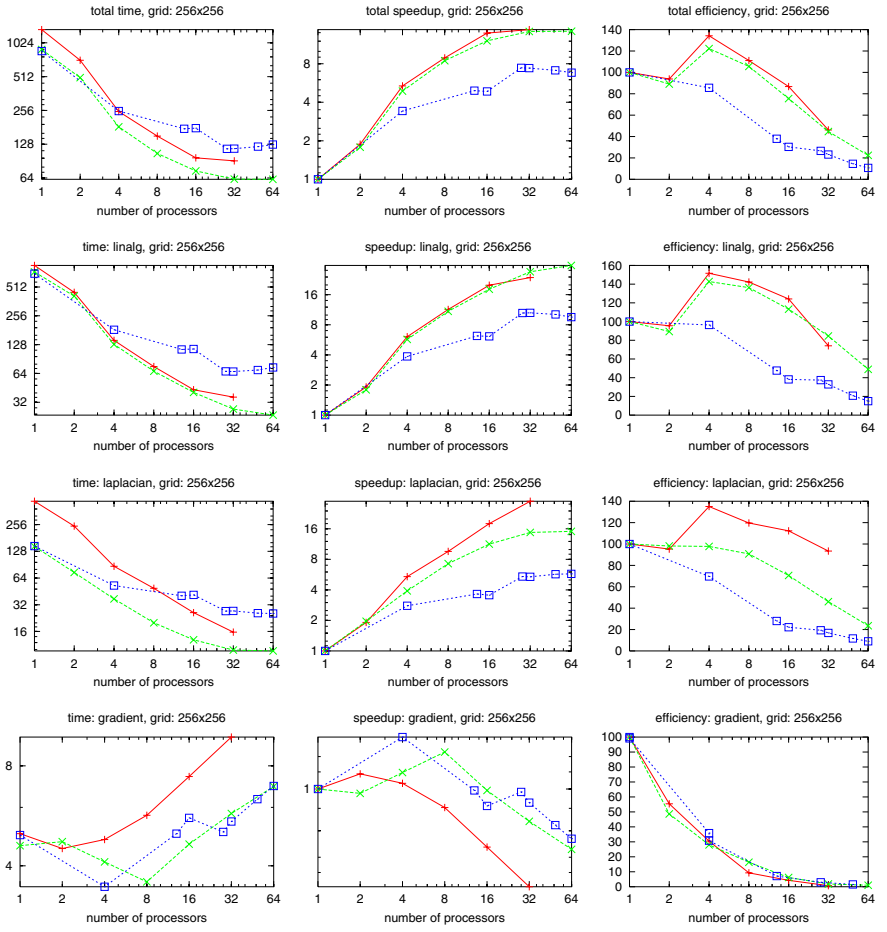

Fig. 10. Results on the 256×256 grid for the three approaches (see Fig. 1): 'element' (+++), 'column' (× × ×), and 'subdomain' (□□□). First row: overall results; second row: results for the vector operations in RKC; third row: results for the Laplacian; last row: results for the gradients. In the first column we give the total CPU time in seconds, for various numbers of processors. The second column shows the corresponding speedup factors and the last column the corresponding efficiencies.

(256×256) this efficiency is possible for up to 16 processors, whereas the coarsest grids limit the number of processors that can be efficiently exploited to 8.

For this particular application this property is not a serious limitation to use a multi-processor machine, since the problem anyhow needs a fine spatial resolution to yield a solution that is sufficiently accurate.

In summary, for the application described in this report, the Teras-computer yields speedup factors up to 50, which are obtained with relatively modest programming effort.

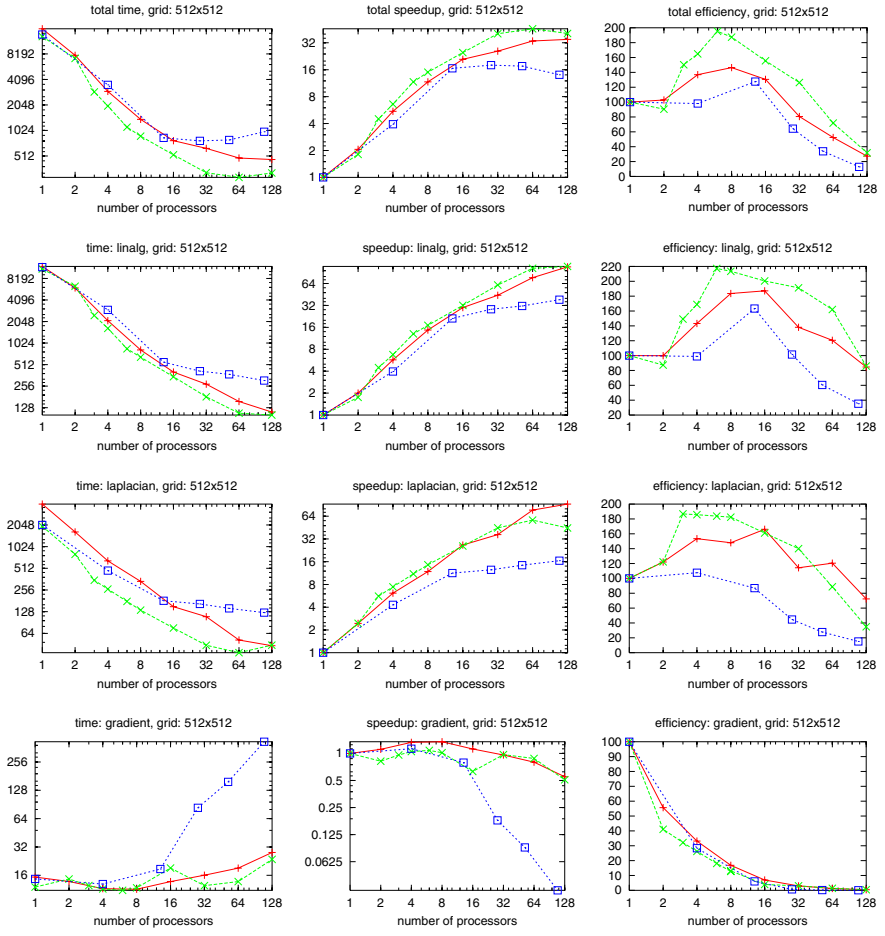

Fig. 11. Results on the 512×512 grid for the three approaches (see Fig. 1): 'element' (+++), 'column' ($\times\times\times$), and 'subdomain' (□□□). First row: overall results; second row: results for the vector operations in RKC; third row: results for the Laplacian; last row: results for the gradients. In the first column we give the total CPU time in seconds, for various numbers of processors. The second column shows the corresponding speedup factors and the last column the corresponding efficiencies.

Acknowledgements

The first author thanks CWI for the warm hospitality during his stay in the autumn of 2001. Furthermore, the authors gratefully acknowledge the referee for the suggestions to improve the paper.

References

[1] J.C. Dallon, Numerical aspects of discrete and continuum hybrid models in cell biology, APNUM 32 (2000) 137–159.

[2] R. Gruber, P. Volgers, A. De Vito, M. Stengel, R.-M. Tran, Parameterisation to tailor commodity clusters to applications, J. Future Generation Comput. Syst. 19 (2003) 111–120.
[3] H.G.E. Hentschel, A. van Ooyen, Models of axon guidance and bundling during development, Proc. R. Soc. Lond. B. 266 (1999) 2231–2238.
[4] B. Lastdrager, Numerical solution of mixed gradient-diffusion equations modelling axon growth, CWI, Report MAS-R0203, 2002.
[5] OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface.
[6] SGI Origin 3000 Teras manual, SARA, Academic Computer Centre, Amsterdam.
[7] B.P. Sommeijer, L.F. Shampine, J.G. Verwer, RKC: an explicit solver for parabolic PDEs, J. Comput. Appl. Math. 88 (1997) 315–326.
[8] J.G. Verwer, B.P. Sommeijer, A numerical study of mixed parabolic-gradient systems, J. Comput. Appl. Math. 132 (2001) 191–210.