Parallel ocean flow computations on a regular and on an irregular grid.

Martin B. van Gijzen*

Abstract

Ocean flow problems can be discretized and solved on a regular grid, by taking land points into account in the computations, or on an irregular grid. In the latter approach, the number of unknowns is less than for the regular grid. The data structures are completely different for the two approaches. As a consequence, different numerical techniques may be required. In this paper we study different preconditioners, based on the EBE-preconditioner that was initially proposed by Hughes et al [2]. We also show how the algorithms can be parallelized and we give results obtained on a cluster of workstations.

1 Introduction.

Accurate simulation of ocean flow requires a high resolution model, involving up to millions of unknowns. For simulations with these models one needs the most powerful parallel computers. The computing power of these machines can only be exploited if the relevant algorithms are well parallelizable.

An important part of the computer time for an ocean flow simulation is spent in the solution of linear systems of equations. Parallelization of solution methods for these systems is not trivial. In this paper we study the parallelization of iterative solution methods, and in particular of the preconditioner. We apply the iterative solution techniques for the solution of the discretized Poisson equation in spherical coordinates on a regular, and on an irregular grid.

The research is part of a larger project. The aim of that project is to develop a parallel code for high resolution ocean flow simulation.

2 The model problem.

Our numerical example is a very simple model for the spreading of pollution from a small source in the Pacific. This problem can be modeled by the Poisson equation with proper right-hand side and boundary conditions. We have expressed the problem in spherical coordinates, with constant radius, to get the results in the familiar longitudes and latitudes.

The Poisson equation in spherical coordinates with a constant radius is given by

$$-\frac{\partial}{\partial\alpha}\frac{1}{\cos^2(\theta)}\frac{\partial u}{\partial\alpha} - \frac{\partial^2 u}{\partial\theta^2} + \tan(\theta)\frac{\partial u}{\partial\theta} = f, \tag{1}$$

*Mathematical Institute, University of Utrecht, P.O. Box 80.010. NL-3508 TA Utrecht, The Netherlands, `E-mail: vangyzen@math.ruu.nl`; The research is funded by NWO, the dutch organization for scientific research.

with $\alpha \in [-\pi, \pi]$ and $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$. For the right-hand side function we have selected $f = -0.4$. We solve this equation on a sphere, and therefore have the cyclic boundary condition

$$u(-\pi, \theta) = u(\pi, \theta). \tag{2}$$

The South Pole is land, and hence is not a part of the domain. The North Pole leads to a singularity because of the spherical coordinates. This problem is solved by excluding latitudes higher than 88°, or in our coordinate system $\theta < 1.54$. Along the coasts, and on the North Pole, we assume homogeneous Neumann boundary conditions.

$$\frac{\partial u}{\partial n} = 0. \tag{3}$$

The source of pollution is modeled by the condition

$$u = 5 \text{ in } [0.45,0.55][-0.5,-0.4] \ . \tag{4}$$

The PDE has been discretized with linear triangular finite elements. We have used a Newton-Cotes numerical integration rule to compute the element matrices and element vectors.

A mesh for triangulation of the oceans and the seas is generated from topographical data. The mesh has been decomposed into equally sized vertical strips. This domain decomposition has been exploited in the parallelization of the computations. The (sea) grid points are equidistantly distributed with a vertical and a horizontal distance of 4°. Figure 1 shows the mesh and its decomposition into two subdomains. The discretization yields a (nonsymmetric)

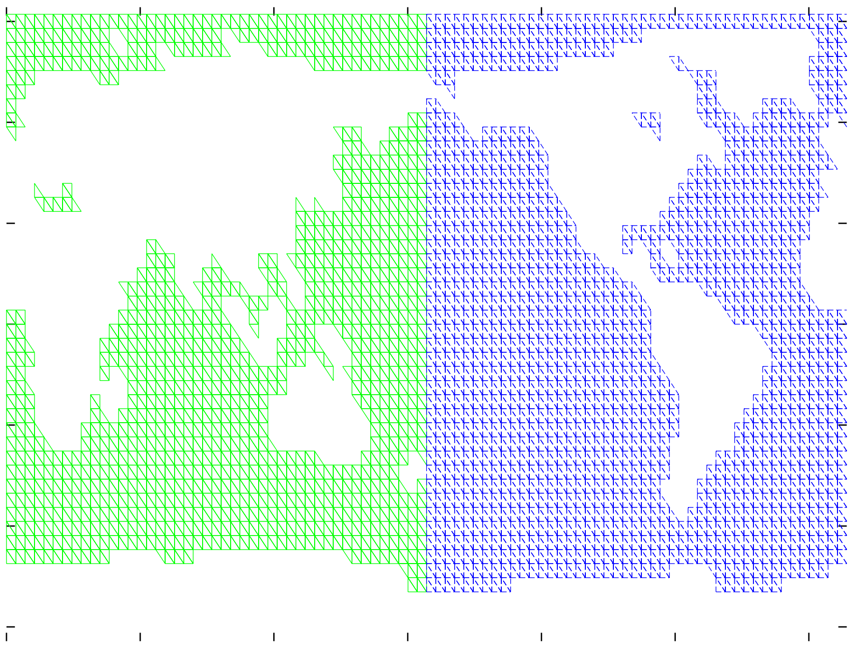

Figure 1: Domain decomposition of a triangulation of the Earth.

linear system

$$Ku = f \quad . \tag{5}$$

An important question is whether the land grid points should be included in this system, by including dummy equations. The consequences of this choice when solving (5) is the subject of the next section.

3 Solution method and parallelization.

Krylov subspace methods like GMRES [5], Bi-CGSTAB [11], and BiCGstab(ℓ) [6] are powerful techniques for solving large and sparse nonsymmetric linear systems of equations. Apart from scalar operations, the methods comprise of inner products, vector updates, matrix-vector multiplications and preconditioning operations. The vector update and inner product operation lead to similar operations for both the regular and the irregular grid approach. The matrix-vector multiplication and the preconditioning operation, however, differ considerably for the two approaches. We will discuss them in some detail here.

In the Finite Element Method, the stiffness matrix is assembled from element matrices K_e. This fact can be exploited, in the construction of the preconditioner as well as in the matrix-vector multiplication. The matrix-vector multiplication can be performed elementwise, only the element matrices are used. The Element-by-Element (EBE) matrix-vector multiplication $Kv = w$ can be described by

$$Kv = \sum_{e=1}^{n_e} K_e v_e = \sum_{e=1}^{n_e} w_e = w \quad . \tag{6}$$

The EBE matrix-vector multiplication requires a considerable amount of indirect addressing. Indirect addressing is unavoidable in irregular grid computations, so in that case this is not a real drawback. However, if the land points are included the grid is regular, and the assembled stiffness matrix $K = \sum_{e=1}^{n_e} K_e$, if properly ordered, has only 7 diagonals with nonzero elements. Operations with this matrix do not require indirect addressing. Hence, regular grid computations are preferably done with the assembled matrix.

With preconditioning we (symbolically) multiply (5) with a suitably chosen matrix P^{-1}:

$$P^{-1}Ku = P^{-1}f \quad . \tag{7}$$

The matrix P is called the preconditioner. It should be an easily invertible approximation for K. A popular way to obtain a preconditioner is to construct it in decomposed form, $P = LU$, with L a lower triangular matrix and U upper triangular.

Hughes et al. [2] have proposed a preconditioner that, apart from a global diagonal matrix D, makes use of element matrices only. They define the preconditioner P in terms of a product of (factors of) element preconditioning matrices P_e:

$$P_e = I + D^{-\frac{1}{2}}(K_e - diag(K_e))D^{-\frac{1}{2}} \quad , D = \sum_{e=1}^{n_e} diag(K_e). \tag{8}$$

For these element matrices a decomposition $P_e = L_e U_e$ is made. The EBE-preconditioner is then defined by the product of element matrices

$$P = D^{\frac{1}{2}} \prod_{e=1}^{n_e} L_e \prod_{e=n_e}^{1} U_e \; D^{\frac{1}{2}} \quad . \tag{9}$$

Note that operations with the inverse of this matrix can easily be carried out, by performing a sequence of forward substitutions on the element matrices L_e, followed by a sequence of back substitutions on the U_e.

The idea of the preconditioner (9) is that the product of element matrices may serve as a suitable approximation for the (scaled) stiffness matrix if K is diagonally dominant. If this is the case the element matrices $L_e - I$ and $U_e - I$ will be small in norm. It can be shown [12] that the error term in the approximation for K consists of products of element matrices of this kind. Hence, if they are small in norm, the error term will be small, which makes a good approximation. In [12] it is also shown that the matrix

$$Q = D^{\frac{1}{2}}(I + \sum_{e=1}^{n_e}(L_e - I))(I + \sum_{e=1}^{n_e}(U_e - I))D^{\frac{1}{2}} \qquad (10)$$

is a better approximation for the scaled stiffness matrix in the sense that the error term in the approximation is smaller than for (9). Note that again the preconditioner is of the form $Q = LU$. This preconditioner is well suited for regular grid computations. The factors L and U have the same nonzero-pattern as the (lower and upper triangular part of the) assembled stiffness matrix.

We have parallelized the matrix-vector multiplication, as well as the inner product and vector update operation, using a domain decomposition approach. In the domain decomposition all elements are uniquely assigned to a subdomain. The subdomains share grid point at the boundaries, and overlap in this sense. The domain decomposition and subsequent parallelization is described in some detail in [13]. Parallelizing the preconditioners is more cumbersome. The approach we have taken is to construct and apply local preconditioners per subdomain, disregarding the fact that the subdomains are coupled. After a back- or forward substitution with these local preconditioners the values in grid points at the boundaries of the subdomains differ, since the subdomains overlap. To solve this problem the values at the boundaries are simply averaged. This idea has initially been proposed in [4]. For more recent, and presumably more effective approaches, see [9].

4 Numerical experiment

Our numerical experiments have been done on a cluster of 6 SUN ELC workstations. Each SUN ELC computer has a SPARC processor with a clock speed of 33 Mhz. The computers are connected via Ethernet with a maximal throughput of 10 Mbit/s. The computations have been performed in single precision arithmetic (≈ 8 significant digits). The communication has been implemented using the Oxford BSP library [3], according to the BSP programming model [1], [10]. We selected the iterative solver BiCGstab(4) [6], with special techniques to improve the convergence properties [8] and the accuracy of the solution [7]. For the preconditioner we have used (9) on the irregular grid, and (10) for the regular grid. The iterative process is stopped if the reduction in the norm of the residual is greater than 10^6.

We have solved the problems using 2, 3, 5, or 6 subdomains. The number of elements and the number of sea grid points may differ per domain. Table 1 gives the maximum and the average number of elements per subdomain in column two and three. The maximum and the average number of sea grid points are tabulated in column four and five. The total number of grid points (land and sea) per subdomain is the same for all subdomains. They are tabulated in column six. Note that one sea grid point corresponds with one unknown on the irregular grid, and one grid point (either land or sea) corresponds with one unknown on the regular grid. The ratio of the maximum number of sea points and the average number of sea points gives a good impression of the load imbalance for the irregular grid approach. Load imbalance can not be neglected for the irregular grid, and one can not expect to gain more than a factor 2.5, the ratio of the maximum number of sea points, varying from 2 to 6

#subdomains	max #elm	average #elm	max #sea points	average #sea points	total #grid points
2	2607	2288	1512	1365	2070
3	1904	1525	1082	914	1395
5	1270	915	723	561	855
6	1085	763	612	473	720

Table 1: Number of elements and grid points per subdomain

processors. We can expect to observe a better scaling for the regular grid approach, since for this approach the load balancing is more or less perfect.

Table 2 lists elapsed times and numbers of matrix-vector multiplications (matvecs) for different numbers of subdomains for both the irregular and the regular grid problem. For BiCGstab(4), 8 matvecs correspond to one iteration. The speed-ups for the irregular grid

# subdomains	irregular grid		regular grid	
	# matvecs	CPU-time	# matvecs	CPU-time
2	600	77.8	352	24.8
3	624	61.8	304	18.2
5	456	39.5	344	23.8
6	496	41.5	352	27.1

Table 2: Matvecs and elapsed times for different numbers of subdomains.

approach are better than for the regular grid approach. This is probably due to the fact that the computation part of the regular grid method is much faster, and hence the communication between processors *relatively* more expensive.

5 Concluding remarks.

The regular grid version clearly outperforms the irregular grid version for our problem, both with respect to numbers of matvecs, as with respect to computing times. The preconditioners for both approaches seem to remain effective when the number of processors is increased, at least for modest numbers of processors.

In our study we have used a coarse grain parallel approach. The preconditioner (10) is not well suited for fine grain parallelism, like for example used in HPF. If we would have used HPF as a method for parallelization then our conclusions might have been completely different. This is a topic of further research in our project. Another topic of research is the generation of a suitable grid. Solving the Poisson equation in spherical coordinates is illustrative, but of coarse not the right approach if one wants to solve huge systems. Solving the Poisson equation with more or less uniformly distributed grid points on the sphere seems a better way.

Acknowledgements. We like to thank Hakan Oksuzoglu for his help with the example. We thank Henk van der Vorst and Gerard Sleijpen for many helpful discussions on numerical details. Diederik Fokkema is greatfully acknowledged for supplying the BiCGstab(ℓ) code. Frank van der Stappen and Rob Bissling are acknowledged for help with the parallelization.

References

[1] R.H. Bisseling Sparse Matrix Computations on Bulk Synchronous Parallel Computers. To appear in: *Proceedings International Conference on Industrial and Applied Mathematics,* Hamburg, July 1995, special issue of ZAMM (1996)

[2] T.J.R. Hughes, I. Levit, and J. Winget. An element-by-element solution algorithm for problems of structural and solid mechanics. *Comput. Methods Appl. Mech. Engrg.* 36:241–254, 1983

[3] R. Miller and J. Reed. The Oxford BSP library users' guide, version 1.0 *Oxford Parallel*, Oxford 1993

[4] G. Radicati di Brozolo and Y. Robert. Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor. *Parallel Comput.*, 11:223–239. 1989

[5] Y. Saad and M.H. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986

[6] G.L.G. Sleijpen and D.R. Fokkema. BiCGstab(ℓ) for linear equations involving matrices with complex spectrum. *ETNA* 1:11–32, 1994

[7] G.L.G. Sleijpen and H.A. van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. To appear in: *Computing*

[8] G.L.G. Sleijpen and H.A. van der Vorst. Maintaining convergence properties of BiCGstab methods in finite precision arithmetic. *Numerical Algorithms*, 10:203–223, 1995

[9] .K.H. Tan. Local coupling in domain decomposition. *Thesis*, Utrecht University, Utrecht (1995)

[10] L. Valiant A bridging model for parallel computation. *Comm. ACM* 33:103–111, 1990

[11] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992

[12] M.B. van Gijzen An analysis of element-by-element preconditioners for nonsymmetric problems. *Comput. Methods Appl. Mech. Engrg.* 105:23-40, 1993

[13] M.B. van Gijzen. Parallel iterative solution methods for linear finite element computations on the Cray T3D. Proceedings of the HPCN Europe 1995 conference. *Lecture Notes in Computer Science*, Springer Verlag, 919:723–728, 1995.

A Solution of the model problem.

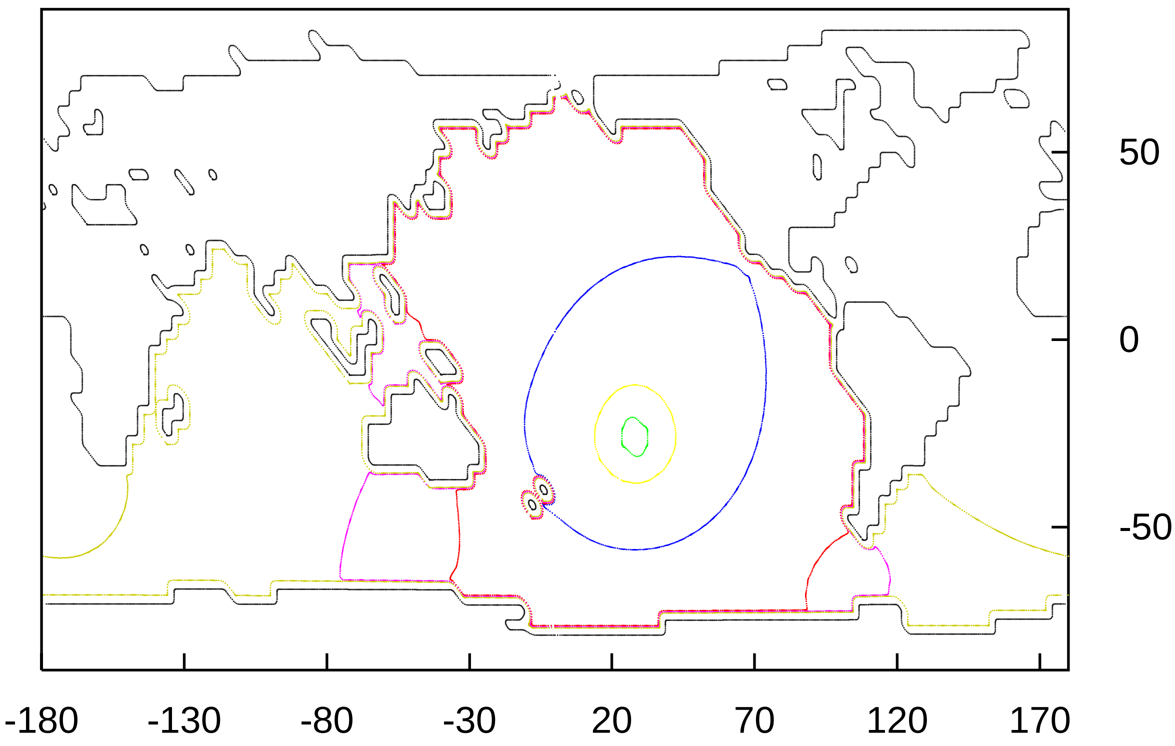

Figure 2: Contour plot of the solution, contour lines at $u = 0$, $u = 2, u = 2.5, \cdots, u = 5$.

B Analysis of the parallel performance with the BSP model.

The program is parallelized according to the Bulk Synchronous Parallel (BSP) programming model. In this model some assumptions are made on the computer. A BSP computer consists of a number of processors, each with its own memory, a communication network that provides access to other processor's memories, and a mechanism for global synchronization [1]. No distinction is made in access time between different remote memories. A cluster of workstations, connected via ethernet can be regarded a BSP computer. A BSP algorithm is composed of *supersteps*. A superstep is either a computation step or a communication step. Each step is terminated by a global synchronization. The separation of communication and synchronization makes it possible use efficient one-sided communication, like remote write ('store') and remote read ('fetch') operations.

The cost of a superstep can be expressed in a cost function, with as unit the time of a floating point operation. Very simple cost functions for a communication and for a computation step are given in [1]. For a communication step we have

$$T_{comm}(h) = hg + l. \tag{11}$$

Here h is the number of reals to store in remote memory or fetch from remote memory, g is the time to fetch or store one real, and l is the cost of synchronization (plus latency). For a computation step we have

$$T_{comp}(w) = w + l. \tag{12}$$

Here w is the maximum amount of floating point operations by any processor in a superstep.

To analyze the cost of the BiCGstab(4) algorithm we have to determine which supersteps are taken in an iteration. The BiCGstab(ℓ) algorithm can be described by

```
Choose an initial guess x_0 and some r̃_0
r_0 = P^{-1}(b - K x_0)
k = 0
u_0 = 0, α = ρ_0 = ω = 1
while ( ||r_k||_2 > ε||r_0||_2 & k < mxmv ) do
    k = k + 2ℓ
    ρ_0 = -ωρ_0
    for j = 0, ℓ-1 do
        ρ_1 = (r_j, r̃_0),    β = α(ρ_1/ρ_0)
        ρ_0 = ρ_1
        for i = 0,...j do,
            u_i = r_i - β u_i
        end
        u_{j+1} = P^{-1} K u_j
        σ = (u_{j+1}, r̃_0),    α = ρ_0/σ
        for i = 0,...j do,
            r_i = r_i - α u_{i+1}
        end
        r_{j+1} = P^{-1} K r_j
        x = x + α u_0
    end
    for i = 1, ℓ do
        for j = 1, i do
            Z(i,j) = Z(j,i) = (r_j, r_i)
        enddo
        γ_i = (r_0, r_i)
    end
    γ = Z^{-1} γ
    ω = γ_ℓ
    for i = 1,...ℓ do
        x = x + γ_i r_{i-1}
        u_0 = u_0 - γ_i u_i
        r_0 = r_0 - γ_i r_i
    end
endwhile
```

This does not include modifications to improve convergence [8] and accuracy [7], but these modification require few operations. To simplify the analysis we only regard the operations on the regular grid. On the regular grid all processors have equal workload, and the (difficult to model) overhead for indirect addressing is less. Accept for scalar operations, the algorithm is composed of four operations: matrix-vector product, preconditioning operations, inner product, and vector update. The matrix-vector multiplication is composed of one computation step and one communication step. The number of floating point operations w is $14n$ with n the number of gridpoints per subdomain. The number of reals to be sent (and received), h, is equal to nb, with nb the number of boundary nodes. This has to be done twice for the preconditioner, one time for the back substitution and one time for the forward substitution. The number of flops w is equal to $18n$. The inner product is composed of two computation and two communication steps. Only $p-1$ reals, p is the number of processors, are received in the first communication step, and $p-1$ reals are sent in the second communication step. In the two computation steps $w = 2n$ flops are performed. The vector update operation requires no communication, and $2n$ flops. Since no communication is performed we can regard the vector update operations to belong to another (computational) superstep, and no synchronization is required. The communications and computations per iteration of the various operations are tabulated in Table 3. Table 3 also gives the numbers of matvecs, preconditioning operations, inner products and vector updates in a BiCGstab(4) iteration. With the results of Table

	matvec	preconditioning	inner product	update
h	nb	$2nb$	$2p-2$	0
super steps	2	4	4	0
w	$14n$	$18n$	$2n$	$2n$
number per iteration	8	8	22	36

Table 3: Computations and communication per iteration

3 we can compose a total cost function for a BiCGstab(4) iteration. The total number of supersteps is 136. The total number of floating point operations w is $372n$, and the total number of reals h to fetch or store is $24nb + 44(p-1)$. For the total cost function we get

$$T_{tot} = 372n + (24nb + 44(p-1))g + 136l \tag{13}$$

The function T_{tot} gives the workload in *flop*. The first term corresponds to computation, the second to communication, and the third to synchronization. To get predictions for the actual computing time we have to multiply T_{tot} by the processor speed s.

We have the following parameters in the model: n, nb, s, l, g, and p. The first two parameters depend on the problem and on the domain decomposition. We have for the number of boundary points $nb = 90$, and for the number of gridpoints per domain $n = 45\times(1+90/p)$. The parameters s, g, and l are system depend. We have performed benchmarks [1] to determine the values. Table 4 gives the results. The processor speed is approximately 1.0 in all measurement and, of course, shows no dependence on the number of processors p. For the parameter g and l we make the crude simplification to assume a linear dependence on the

[1] Benchmark code supplied by Rob Bisseling of the University of Utrecht

p	s	g	l
2	1.01	85.2	4918
3	1.01	130.8	4452
5	1.00	156.3	6016
6	1.00	178.3	8659

Table 4: Measurement of system dependent parameters.

number of processors. We get the linear least squares curves $g = 53+21p$ and $l = 2400+900p$. Combining all relations we get the following cost function

$$T_{tot} = 0.5 + \frac{1.5}{p} + 0.17p + 10^{-3}p^2 \quad . \tag{14}$$

This cost function measures in $Mflop$, but since the processor speed is approximately $1Mflop/s$, it also gives the prediction of the actual time in s for an iteration. Figure 3 shows this cost function and the actual measurements. The actual time per iteration can be computed from the data in Table 2. The number of iterations is the number of matvecs divided by 8. The

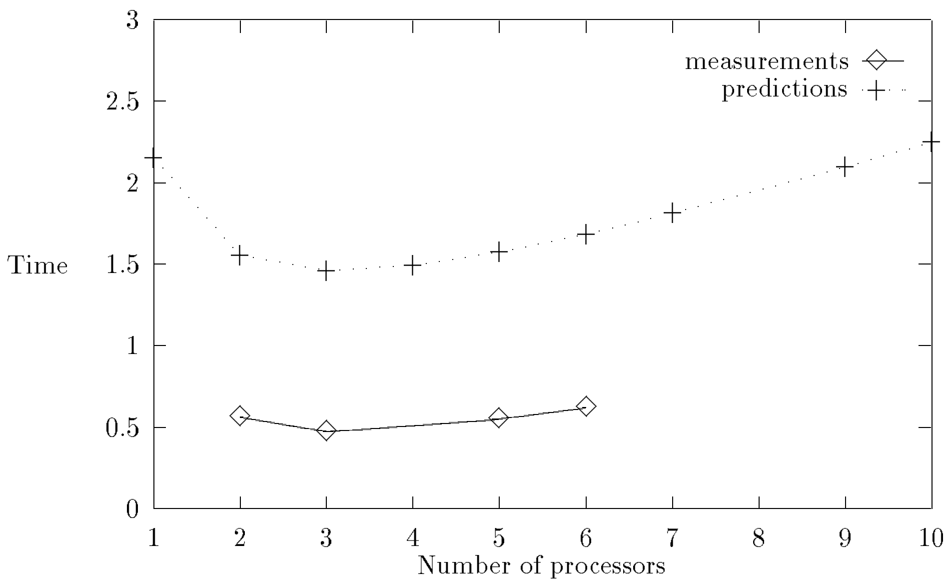

Figure 3: Measured and predicted times per iteration.

predictions are too high. This can (partly) be explained by how the benchmark is carried out. The processor speed is determined by performing vector-update operations. But for example an inner product operation can be performed more efficiently. Hence the processor is estimated too low for the analysis as a whole. The parameter l is estimated for messages of length larger than 1. The inner product operation requires communication with messages of lenght 1. Because of this, l is estimated too high for our analysis. Although the predicted times are too high, the qualitative behaviour is described well by the BSP-model.