Parallel Computing 19 (1993) 651–666
North-Holland

PARCO 749

A parallel chopping algorithm for ODE boundary value problems

Marcin Paprzycki [a] and Ian Gladwell [b]

[a] *Department of Mathematics and Computer Science, University of Texas of the Permian Basin, Odessa, TX 79762, USA*
[b] *Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA*

Received 1 May 1990
Revised 28 October 1990, 24 February 1992, 15 June 1992

Abstract

Paprzycki, M. and I. Gladwell, A parallel chopping algorithm for ODE boundary value problems, Parallel Computing 19 (1993) 651–666.

A chopping algorithm proposed by Russell and Christianssen [14] for the solution of linear boundary value problems (BVPs) in ordinary differential equations (ODEs) is extended. On a computer with P processors, at each stage the BVP is solved numerically on P meshes using a code based on COLNEW [1,4]. The solutions from the P processors are combined in an algorithm which chops the range of integration and poses a set of new, 'simpler' BVPs on subintervals. These are treated recursively. Experiments on three problems with boundary and turning layers show that this algorithm can be more efficient than using COLNEW directly, and without a significant loss of accuracy. It may also be used as a mesh selector for COLNEW.

Keywords. Linear boundary value problems; ordinary differential equations; multiprocessor systems; chopping algorithm.

1. Introduction

A number of papers have recently considered parallelizing the numerical solution of two-point boundary value problems (BVPs) for ordinary differential equations (ODEs). Most address the parallelization of the linear algebra arising in the BVP solvers, see [2,12,13]. As shown in [7] and [8], for a large group of typical BVPs the linear algebra takes between 25% and 50% of the overall time of finding the solution. Most of the remaining parts of the BVP solver are either parallelizable or require almost no computing time [5,11].

Discretization of the BVP leads to linear systems of almost block diagonal (ABD) structure. The proposed approaches to BVP solution can be classified according to the granularity of parallelization they represent. Wright and Pereyra [15] designed an algorithm for solving ABDs on vector computers, but used an Alliant parallel computer which performed automatic fine-grained micro-parallelization. A medium granularity of parallelization is represented by the block Gaussian elimination algorithm based on the level 3 BLAS [6] as presented in [13]. A higher, but still medium, granularity of parallelization is represented by treating ABDs as block tridiagonal. Ascher and Chan [2] and others utilize an odd-even

Correspondence to: I. Gladwell, Department of Mathematics and Computer Science, University of Texas of the Permian Basin, Odessa, TX 79762, USA.

reduction to solve such a system. Paprzycki and Gladwell [12] exploit the ABD structure in a tearing algorithm that seems to be stable. This approach yields speedups of about 4 for systems as large as 480 internal blocks when using 20 processors.

The aim of this paper is to explore the possibility of solving a two-point linear BVP for ODEs using a 'whole problem' level of granularity. The BVP considered is

$$\underline{y}' = A(x)\underline{y} + \underline{q}(x), \quad a \le x \le b, \qquad (1)$$

$$B_a\underline{y}(a) + B_b\underline{y}(b) = \underline{\beta}.$$

The method proposed is similar to domain decomposition and is based on 'chopping' [14].

Section 2 recalls a simple sequential chopping algorithm. Section 3 shows how to implement this in a multiprocessor environment. Section 4 shows that, under certain quite realistic assumptions, the chopping procedure leads to at worst linear error growth. Finally, Section 5 discusses numerical implementation. The new algorithm can give a speedup over COLNEW [1,4] run sequentially. COLNEW is a spline collocation code for solving nonlinear ODE BVPs. It uses a modified Newton method, an efficient adaptive mesh placement scheme and a monomial spline basis. It employs (potentially parallel) internal condensation to reduce the number of variables. This code is very highly regarded and has been used as the basis for applications packages.

2. Sequential chopping

The basic ideas are briefly presented in [14]. Assume the BVP is to be solved over an interval $[a, b]$ with a given tolerance TOL. Then, the 'chopping' algorithm can be described (see *Fig. 1*):

(1) An approximate solution and error estimates are calculated for a given discretization on $[a, b]$.

(2) Suppose that on subinterval $[x_1, x_2] \subset [a, b]$ the estimated error is smaller than TOL, and $[x_1, x_2]$ cannot be extended contiguously and this relation remain true. Then, the solution on $[x_1, x_2]$ is accepted. If there are no accepted subintervals, halve the mesh and go back to (1).

(3) A set of BVPs is formulated on those subintervals of $[a, b]$ where the approximation to $y(x)$ has not been accepted. (In *Fig. 1*, new BVPs will be formulated on $[a, x_1]$ and $[x_2, b]$.) For each subinterval the system of ODEs will be solved with new (computed) boundary conditions (BCs).

(4) The process is repeated from (1), independently, for each subproblem created in (3).

According to [14] this procedure may lead to considerable savings in work by reducing the sizes of the numerical problems solved. By reducing the numbers of mesh points, chopping may significantly lower the size of the linear systems (usually the most expensive part of solving the BVP, [8]). Most experiments reported in [14] allowed chopping only at the interval ends. However, the authors claim some success when allowing chopping elsewhere.

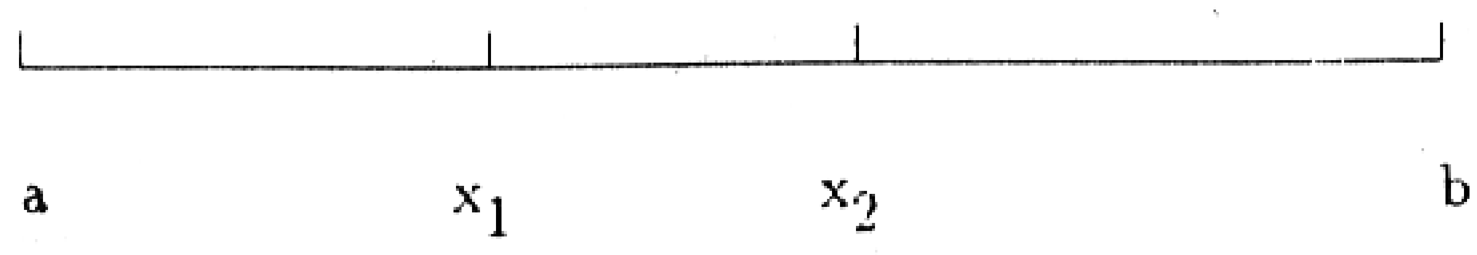

Fig. 1. Subinterval $[x_1, x_2]$ is characterized by a small error. The solution on intervals $[a, x_1]$ and $[x_2, b]$ will be refined independently on new meshes with BCs recalculated at x_1 and x_2.

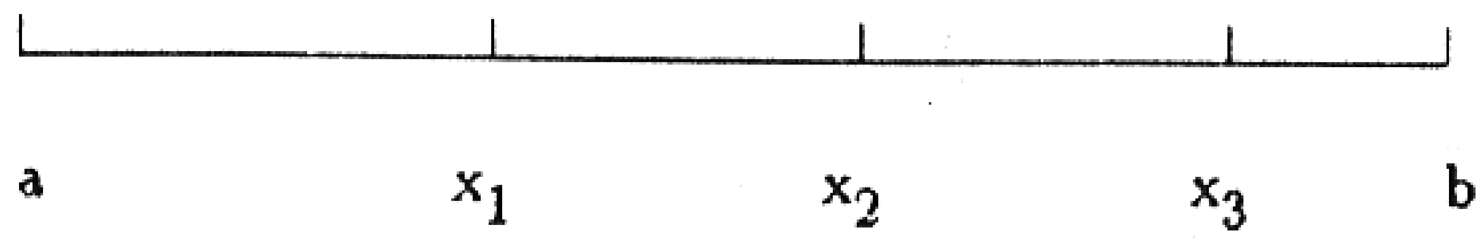

Fig. 2. A problem with a right boundary layer. The sequence of intervals on which refinements are performed is $[x_1, b], [x_2, b], [x_3, b]$.

The general chopping procedure, with chopping allowed anywhere, leads naturally to large granularity parallelization. For example, if (as is assumed above) after the first step of the process the interval $[a, b]$ can be divided into subintervals where solution refinement is required, then each subproblem can be solved independently using a general BVP solver.

If there are large differences in difficulty of solving the problem on the subintervals, even if similar meshes are used to start the calculations on each subinterval, it may take different amounts of work on each processor before either additional chopping is performed or the solution is accepted. Hence if each subinterval is assigned to a processor, simple chopping may lead to large load imbalances and synchronization problems.

The degree of parallelization is problem dependent. Problems with just left or right boundary layers and otherwise smooth solutions will be almost impossible to parallelize because at each stage, there will usually be a division into just one accepted and one unaccepted part (see *Fig. 2*).

3. Parallel chopping

To parallelize the chopping procedure, a more balanced way to split the interval into 'good' and 'bad' parts is needed. One possible approach, whilst keeping a large granularity of parallelization, is to use several processors to compute information that will permit determination of the accepted and unaccepted subintervals. One way to achieve this follows:

(a) on each interval solve the BVP on different meshes in parallel,
(b) use information from the different meshes to find the parts of the interval that are to be accepted,
(c) reformulate the BCs of the BVP for the unaccepted subintervals,
(d) repeat from (a) independently for each unaccepted subinterval.

The linear BVP (1) is solved using a general BVP solver. A set of meshes with different densities of mesh points in different subintervals is used. The problem is solved on each mesh and the results compared, permitting the decision either that the problem is solved or where to chop.

To investigate properly the shape of the solution, the distribution of mesh points is exemplified in *Fig. 3*. The number of meshes depends on the number of active processors, P.

Points common to all meshes are *main* mesh points. Each main mesh point is at the center of one high density region of mesh points. This mesh point distribution is chosen to prevent loss of information about boundary layers and turning layers. The regions of high mesh density cover the whole interval and overlap. This is designed to detect the location of turning layers. The first and last meshes are designed to improve the chances of detecting boundary layers.

The error is estimated by comparison of the results obtained on a 'single' and on a 'double' mesh (see [3], ch IX). The solutions on the meshes and their doubled counterparts are calculated. Then error estimates are computed at all mesh points in all meshes. This information is used to:

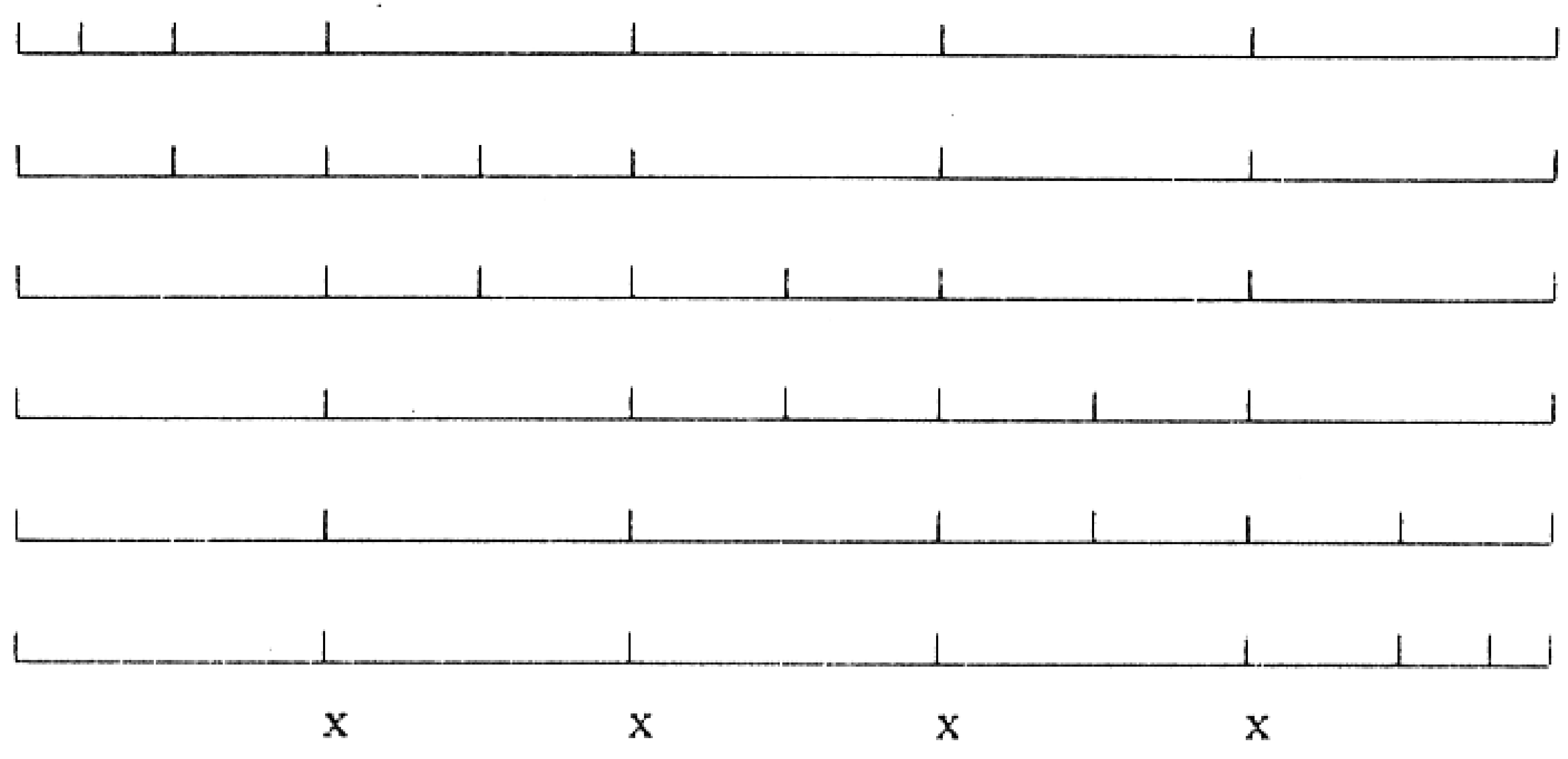
Fig. 3. A set of $P = 6$ starting meshes. x marks the position of the *main* mesh points.

(i) quit the problem if the tolerances are satisfied by the solution on any single mesh or throughout the main mesh points on the set of meshes,
(ii) make decisions about where to chop.

When the determining where the chop, the considerations include:

(A) The procedure is based on local error estimates. These estimates are based on comparing solutions calculated on 'single' and 'double' meshes. Possibly neither represents the true solution well. Then the error estimates may indicate little about the relation between the calculated and true solutions. In such cases it is assumed the error estimates will be large, i.e. the program will not be deceived.
(B) Error estimates can sometimes be small solely due to sign changes, so more than one piece of favorable evidence is required before chopping.

These observations lead to the adoption of the following criteria:

(I) Chop only on main mesh points. (Because they are in the subintervals with maximum density of mesh points, the error estimates there should be the most reliable.)
(II) For each subinterval between main mesh points, take the maximum norm over all error estimates. To chop, this norm and the error estimate at the main mesh point must both be smaller than the assumed tolerance. (This is designed to prevent the solver losing information about large jumps in the errors in the neighborhood of the main mesh points. Such jumps imply the results at the main mesh points are inaccurate.)
(III) For each accepted main mesh point, the absolute difference between corresponding components of the solution from two successive meshes (where the high density intervals overlap) must be smaller than the tolerance. (This criterion ensures the solution is accepted only when it behaves in a uniform way.)

When criteria (I)–(III) are satisfied at a main mesh point, it is assumed that an accurate approximate solution is available and it can be used to determine new boundary values. Subintervals between those main mesh points chosen as chopping points are accepted. An accurate approximate solution is available throughout these subintervals so they can be removed from the problem.

The BCs at the chopping points are determined using B_a and B_b from the BVP (1) and the calculated values of the solution at the chopping points. The procedure is to substitute the solution at the chopping points to determine new right hand sides for the BCs. Assuming the original problem is well-posed, this procedure produces a new set of well-posed problems. The precise assumptions and a partial analysis of the effects of this approach on the accuracy of the solution is presented in Section 4.

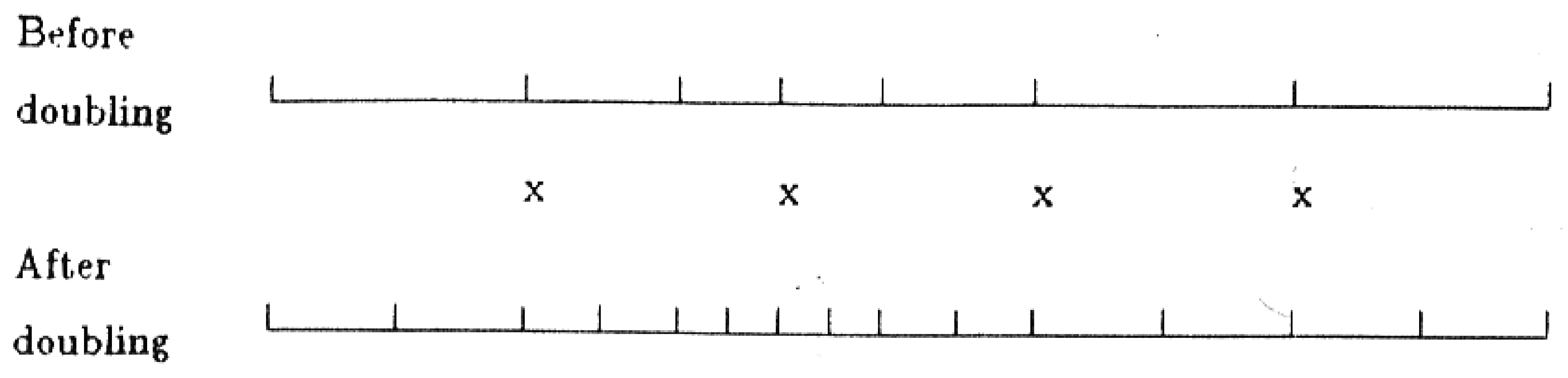

Fig. 4. Mesh before and after doubling; x marks the main mesh points.

When any one of criteria (I)–(III) is not satisfied for any main mesh point, the mesh is doubled. The solution must be recalculated only on the newly doubled meshes to compute the error estimates because the previously calculated double mesh solutions can be reused as the new 'single mesh' solution. The algorithm does not increase the number of main mesh points when doubling; it chops in the same set of main mesh points as in the previous step. If there is again no chopping point, the procedure is repeated. This is an obvious algorithmic weakness. But, as seen later, this does not affect efficiency when using a large number of processors. *Figure 4* presents one of the meshes from *Fig. 2* before and after doubling.

After chopping, a set of subintervals remains on which we must recalculate the solution. The mesh points in these subintervals were characterized by large and/or unreliable error estimates. Hence, the solution at these points is not reusable. So, the whole procedure is repeated by generating the same set of meshes as in the first step, transformed to these smaller subintervals.

When the problem has been reduced to solving the reformulated BVPs on a set of subintervals, a subset of the processors could be assigned to each subinterval. If the load could be balanced, then it might be possible to speed up the overall calculation. However, in the algorithm above, the number of main mesh points used is directly proportional to the number of processors assigned to the subinterval. The effect of the trade-off between being able to solve on several subintervals simultaneously and decreasing the number of main mesh points per subintervals is unclear. In the examples below, for the most part only one subinterval would be active and these considerations are secondary.

4. Accuracy

Suppose that chopping is performed at $N-1$ points, $z_1, z_2, \ldots, z_{N-1}$, so $[a, b]$ is divided into N subintervals with $z_0 = a$ and $z_N = b$. It will suffice to analyze a typical subinterval. On $[z_i, z_{i+1}]$ the solution $\underline{y}(x)$ of (1) is also the solution of

$$\underline{u}' = A(x)\underline{u} + \underline{q}(x), \quad z_i \le x \le z_{i+1}, \tag{2}$$
$$B_a\underline{u}(z_i) + B_b\underline{u}(z_{i+1}) = \underline{\beta}',$$

where $\underline{\beta}'$ is defined by

$$B_a\underline{y}(z_i) + B_b\underline{y}(z_{i+1}) = \underline{\beta}'. \tag{3}$$

The values $\underline{y}(z_i)$, $\underline{y}(z_{i+1})$ needed to define $\underline{\beta}'$ are unknown. Suppose there are available computed approximations $\underline{w}(z_i)$, $\underline{w}(z_{i+1})$ and they are used to define $\underline{\beta}''$ by

$$B_a\underline{w}(z_i) + B_b\underline{w}(z_{i+1}) = \underline{\beta}''. \tag{4}$$

The problem that is to be solved after chopping is

$$\underline{s}' = A(x)\underline{s} + \underline{q}(x), \quad z_i \le x \le z_{i+1}, \tag{5}$$
$$B_a\underline{s}(z_i) + B_b\underline{s}(z_{i+1}) = \underline{\beta}''.$$

A bound on $\underline{s}(x) - \underline{y}(x)$ due to the BC perturbation $\delta\underline{\beta} = \underline{\beta}'$ is needed.

In [3, p.112] a bound on the effect of perturbing the BCs is developed. If

$$\max(\| B_a \|_\infty, \| B_b \|_\infty) \le 1, \tag{6}$$

then

$$\max_{z_i < x < z_{i+1}} \| \underline{s}(x) - \underline{y}(x) \|_\infty < K \| \delta\underline{\beta} \|_\infty \tag{7}$$

where

$$K = \max \| Y(x)(Q(x_1, x_2))^{-1} \|_\infty \tag{8}$$

and the maximum is taken over all x, x_1, x_2 such that $a \le x_1 \le x \le x_2 \le b$. Here Y is a fundamental solution of the ODE (2) and

$$Q(x_1, x_2) = B_aY(x_1) + B_bY(x_2). \tag{9}$$

Requiring to be K bounded is equivalent to requiring that the BVP is well-posed on all subintervals of $[a, b]$, [3]. Then

$$\beta'' - \beta' = B_a\big(\underline{w}(z_i) - \underline{y}(z_i)\big) + B_b\big(\underline{w}(z_{i+1}) - \underline{y}(z_{i+1})\big). \tag{10}$$

Using (6), for any $x \in [z_i, z_{i+1}]$,

$$\| \underline{s}(x) - \underline{y}(x) \|_\infty < K\big(\| \underline{w}(z_i) - \underline{y}(z_i) \|_\infty + \| \underline{w}(z_{i+1}) - \underline{y}(z_{i+1}) \|_\infty \big). \tag{11}$$

Suppose a tolerance TOL was specified for computing of $\underline{w}(x)$ and that the values $\underline{w}(z_i)$ satisfy

$$\| \underline{w}(z_i) - \underline{y}(z_i) \|_\infty \le c\text{TOL} \tag{12}$$

for a suitable constant $c \sim \mathrm{O}(1)$. Suppose also that in solving for $s(x)$ on $[z_i, z_{i+1}]$, an approximation $\underline{p}(x)$ is calculated such that

$$\| \underline{p}(x) - \underline{s}(x) \|_\infty \le c\text{TOL}. \tag{13}$$

Then, for any $x \in [z_i, z_{i+1}]$

$$\| \underline{p}(x) - \underline{y}(x) \|_\infty \le \| \underline{p}(x) - \underline{s}(x) \|_\infty + \| \underline{s}(x) - \underline{y}(x) \|_\infty \le (1 + 2K)\ c\text{TOL}. \tag{14}$$

Suppose now that $[z_i, z_{i+1}]$ is to be further subdivided, into subintervals $[t_j, t_{j+1}]$ with $t_0 = z_i$ and $t_M = z_{i+1}$. The analysis is a repetition of the preceding. Given the solution $s(x)$ of (5), a BVP is defined on $[t_j, t_{j+1}]$ by

$$\underline{u}'' = A(x)\underline{u} + \underline{q}(x), \quad t_j < x < t_{j+1}, \tag{15}$$
$$B_a\underline{u}(t_j) + B_b\underline{u}(t_{j+1}) = \underline{\gamma}',$$

where $\underline{\gamma}'$ is defined by

$$B_a\underline{s}(t_j) + B_b\underline{s}(t_{j+1}) = \underline{\gamma}'. \tag{16}$$

Because the values $\underline{s}(t_j)$, $\underline{s}(t_{j+1})$ are unknown,

$$B_a \underline{p}(t_j) + B_b \underline{p}(t_{j+1}) = \underline{\gamma}'' \tag{17}$$

is used instead of (16). The computational problem is then

$$\underline{v}' = A(x)\underline{v} + \underline{q}(x), \quad t_j < x < t_{j+1}, \tag{18}$$
$$B_a \underline{v}(t_j) + B_b \underline{v}(t_{j+1}) = \underline{\gamma}''.$$

A bound on $\underline{v}(x) - \underline{s}(x)$ due to the perturbation $\underline{\gamma}'' - \underline{\gamma}'$ of the BCs is needed. If

$$\| \underline{p}(t_j) - \underline{s}(t_j) \|_\infty \le c\text{TOL}, \tag{19}$$

then

$$\| \underline{v}(x) - \underline{s}(x) \|_\infty \le 2Kc\text{TOL}. \tag{20}$$

If the error in approximating $\underline{v}(x)$ by $\underline{\tilde{p}}(x)$ is also bounded by

$$\| \underline{\tilde{p}}(x) - \underline{v}(x) \|_\infty \le c\text{TOL}, \tag{21}$$

then, for any $x \in [t_j, t_{j+1}]$,

$$\| \underline{\tilde{p}}(x) - \underline{s}(x) \|_\infty \le \| \underline{\tilde{p}}(x) - \underline{v}(x) \|_\infty + \| \underline{v}(x) - \underline{s}(x) \|_\infty \le (1+2K)c\text{TOL}. \tag{22}$$

Hence

$$\| \underline{\tilde{p}}(x) - \underline{y}(x) \|_\infty \le \| \underline{\tilde{p}}(x) - \underline{s}(x) \|_\infty + \| \underline{s}(x) - \underline{y}(x) \|_\infty \le (2+4K)c\text{TOL}. \tag{23}$$

For L consecutive choppings this result generalizes to

$$\| \underline{\hat{p}}(x) - \underline{y}(x) \|_\infty \le (1+2K)Lc\text{TOL} \tag{24}$$

where $\underline{\hat{p}}$ is the computed solution on the smallest subinterval. Hence, error growth is at worst linear.

5. Experimental results

5.1. Implementation details

The algorithm above was coded using COLNEW [1,4], since it is a state-of-the-art, efficient sequential BVP solver. The computations were performed on a 20 processor Sequent Symmetry shared memory machine at Southern Methodist University. One processor runs the operating system and algorithm was coded using a master/slave relationship. Hence 18 processors (slaves) were available. FORCE directives, [10], were used to express this master/slave relationship.

There are some points where global decisions must be made. For example, the decision about chopping needs error estimates on all meshes. Synchronization of processes is required before these decisions can be made. The master processor had the responsibility for coordinating the slave processors and making all decisions. In one step of the algorithm each slave processor calculates the solution on a single and on a double mesh (when no doubling occurs) or just on a doubled mesh (when doubling occurs in the preceding step). The opportunities for further parallelization have been neglected in the calculations on single and double meshes.

Table 1
Results for Equation (27)

P	Pts	Stp	Mxd	Ncd
4	35	47	3	12
5	115	39	3	12
6	129	38	3	12
7	93	31	3	12
8	104	36	2	12
9	110	28	2	11
10	99	29	2	11
11	72	23	2	12
12	112	23	2	11
13	121	23	2	12
14	84	20	2	12
15	170	29	2	11
16	113	22	2	11
17	152	22	1	11
18	140	17	1	12
COLNEW	372	11		11

5.2. Practical results

Experiments were performed with three problems from [9]:

$$\epsilon y'' - y' = 0, \quad 0 < x < 1,$$
$$y(0) = 2, \quad y(1) = 5, \tag{25}$$

is characterized by a boundary layer of thickness $O(\epsilon)$ on the right;

$$\epsilon y'' - y = 0, \quad 0 < x < 1,$$
$$y(0) = 20, \quad y(1) = 5, \tag{26}$$

Table 2
Results for Equation (28)

P	Pts	Stp	Mxd	Ncd
4	67	61	3	11
5	70	41	3	11
6	84	42	2	10
7	87	34	2	10
8	87	35	2	9
9	161	30	2	10
10	142	26	1	10
11	192	29	1	10
12	269	25	1	10
13	165	21	1	10
14	176	20	1	10
15	190	20	1	10
16	280	21	1	10
17	208	19	0	10
18	262	17	0	10
COLNEW	1062	23		11

Table 3
Results for Equation (29)

P	Pts	Stp	Mxd	Ncd
7	41	27	3	12
8	44	25	3	12
9	106	23	3	11
10	35	13	3	12
11	107	25	2	11
12	98	21	2	11
13	79	17	2	11
14	108	17	2	11
15	77	13	2	12
16	78	14	2	12
17	57	13	2	12
18	55	11	2	12
COLNEW	944	24		14

is characterized by a boundary layer of thickness $O(\sqrt{\epsilon})$ on the right and a growing solution on the left;

$$\begin{aligned} &\epsilon y'' + xy = 0 \quad -1 < x < 1, \\ &y(-1) = 2, \quad y(1) = 5, \end{aligned} \tag{27}$$

is characterized by a shock (internal) layer of thickness $O(\sqrt{\epsilon})$ near $x = 0$.

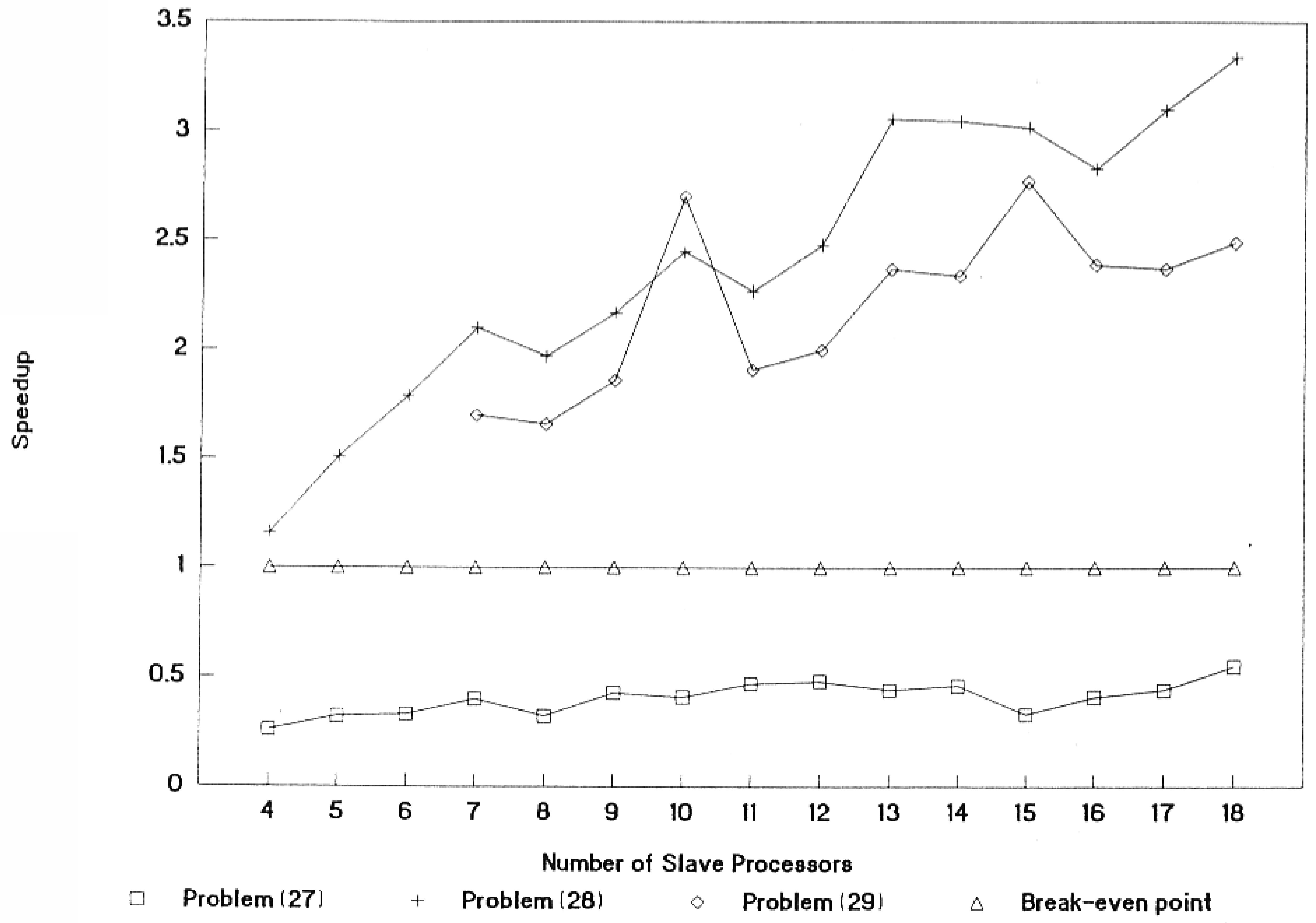

Fig. 5. Speedup.

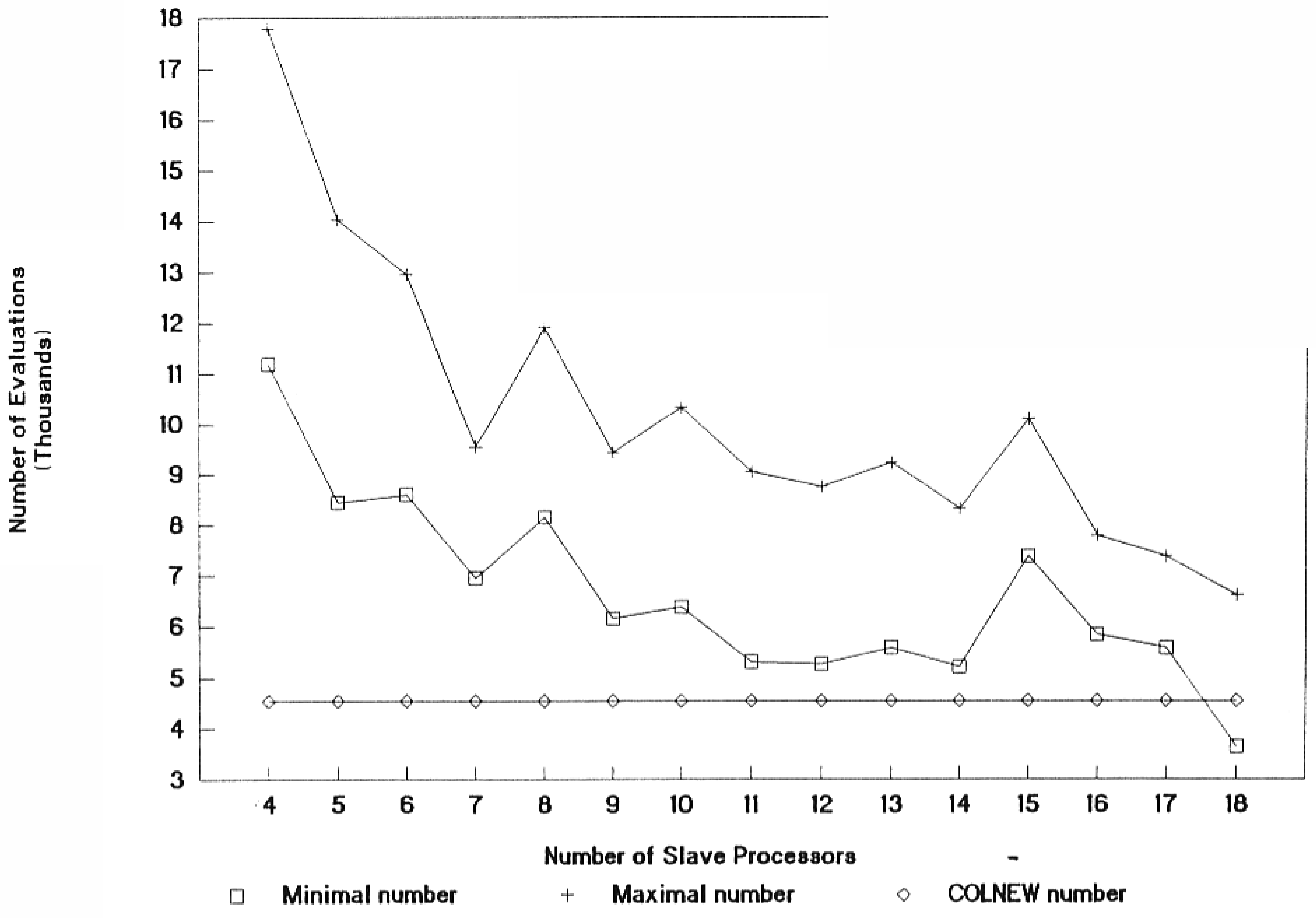

Fig. 6. Function evaluations per processor (Problem 27).

For P active (slave) processors, $P-1$ main mesh points and $P+1$ meshes were used (see *Fig. 3*) Four additional points were used in the high density subintervals. This means that the ratio of the length of the subinterval to the maximum sublength in a high density subinterval is 4:1. For both parallel and sequential programs, 3 collocation points per subinterval were used. That is COLNEW was called with parameters chosen so that the differential equation is satisfied *exactly* at 3 points per subinterval. The sequential program was started with the 'default' 5 equidistributed mesh points.

All problems were solved for $\mathrm{TOL}=10^{-8}$, imposed on the solution and its derivative. For Equations (25) and (27) the solution was calculated with $\epsilon=10^{-4}$ and for (26) with $\epsilon=10^{-5}$. The results are summarized in *Tables 1, 2, 3* for (25), (26), (27) respectively. In these tables:

P	– number of slave processors used,
Pts	– number of points in the final mesh,
Stp	– number of steps to achieve the solution,
Mxd	– maximum number of doublings performed anywhere in the interval,
Ncd	– minimum number of correct digits in the solution across the interval,

Figure 5 shows the speedups for each of the three problems. *Figures 6–11* show the number of derivative evaluations and number of blocks in the linear algebra. In all cases, the corresponding values for COLNEW are plotted. The quantities plotted are:

$\mathrm{Nf}_{\min}$	– minimal total number of function (or derivative) evaluations on a single processor,
$\mathrm{Nf}_{\max}$	– maximal total number of function (or derivative) evaluations on a single processor,
$\mathrm{Nb}_{\min}$	– minimal total number of blocks in the linear algebra on a single processor,

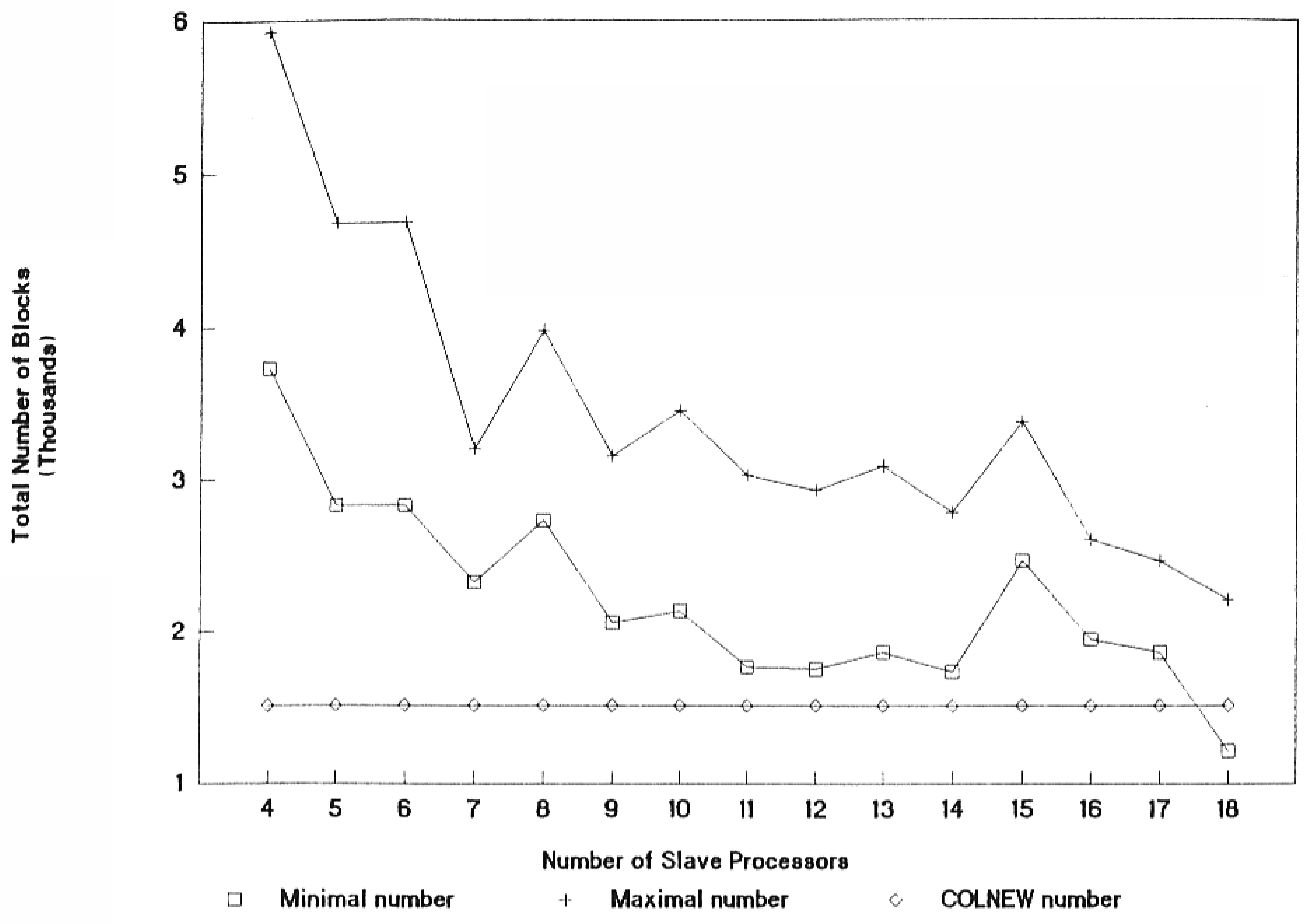

Fig. 7. Blocks used per processor (Problem 27).

Nb_{max}	– maximal total number of blocks in the linear algebra on a single processor,
Nf	– total number of function (or derivative) evaluations in COLNEW
Nb	– total number of blocks in the linear algebra by COLNEW,
Speedup	– (wall-clock time for COLNEW)/(wall-clock time for the parallel algorithm).

In *Table 3*, for $P < 7$ processors, the parallel algorithm failed to compute the solution due to exceeding the memory provided. Given more memory it would have succeeded in each case.

5.3. Analysis of the results

There is a significant difference between the behavior of COLNEW and the program developed here. For Equation (25) the number of mesh points used in COLNEW was moderate. To satisfy the accuracy requirements COLNEW redistributes the mesh rather than increases the number of mesh points. Most points in the final mesh are concentrated at the right end of the interval in the boundary layer. The parallel program is not able to cope with this problem as well as COLNEW. Even for a large number of points in a starting mesh, 23, the program is not able to approximate the rapidly changing solution. The large value of Stp (≥ 17) suggests that frequent doubling was necessary, and, even after doubling, only small parts of the subinterval were accepted. Comparing Nb_{max} with Nb from COLNEW shows clearly that if a large system of equations similar to (25) is to be solved and/or more collocation points are used (so that the size of each block in the discretization increases) the performance of COLNEW would be even more advantageous. Similarly, comparing Nf_{max} with Nf from COLNEW shows that if, each function and derivative evaluation was time

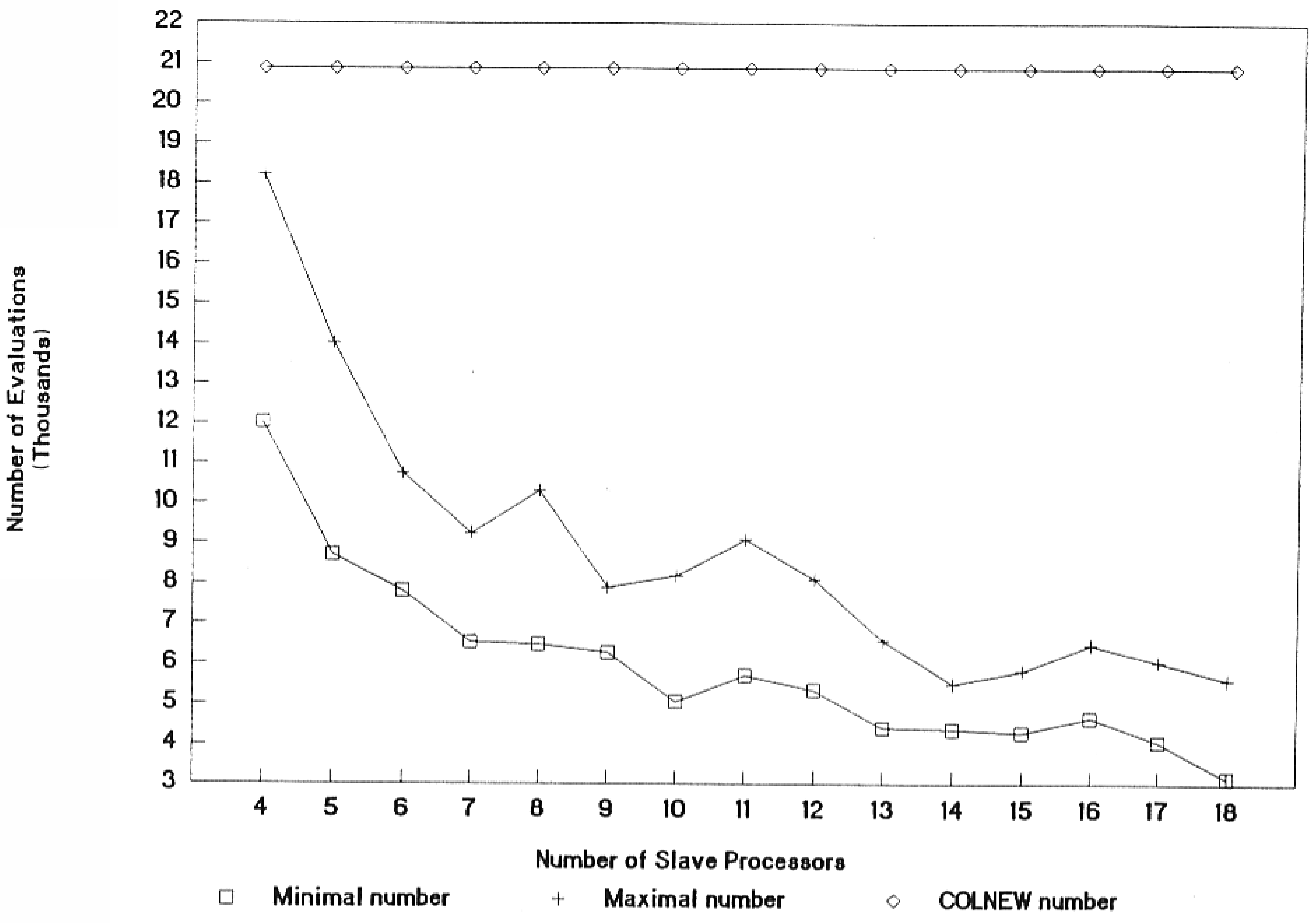

Fig. 8. Function evaluations per processor (Problem 28).

consuming, then the comparative performance of COLNEW would again be even more favorable.

When solving (26), COLNEW not only redistributes the mesh points, but also considerably increases the number of points in the mesh. Based on the number of steps performed, Stp = 23, it seems COLNEW encounters problems when redistributing mesh points in *both* directions. The size of the final mesh was 1026 and the overall number of blocks was almost 7000. The costs of the linear algebra in COLNEW play an important role. The parallel program is able to chop the 'easy' middle past of the solution quickly and then work near the boundaries. Because the boundary layer is of thickness $O(\sqrt{\epsilon})$, the solution changes less rapidly than for (25) and is easy to approximate. The maximum number of blocks used in the linear algebra by any processor was 2 to 3 times smaller than for COLNEW.

Similar remarks apply when solving (27), whose relative difficulty can be clearly observed from the fact that the program was not able to start chopping for $P < 7$. To approximate the fast change in the second component of the solution, $y'(x)$, COLNEW not only redistributed but also considerably increased the size of the mesh. As a result a very high accuracy final solution $y(x)$ was achieved (Ncd = 14). Again, the cost of the linear algebra played an important role. Comparing Nb_{max} with Nb from COLNEW shows clearly that if a large system of equations similar to (26) or (27) is solved and/or a large number of collocation points is used, the performance of the parallel program would be even more advantageous. Similarly, comparing of Nf_{max} with Nf from COLNEW shows that if each function and derivative evaluation is time consuming, then the performance of the parallel program would be even more favorable.

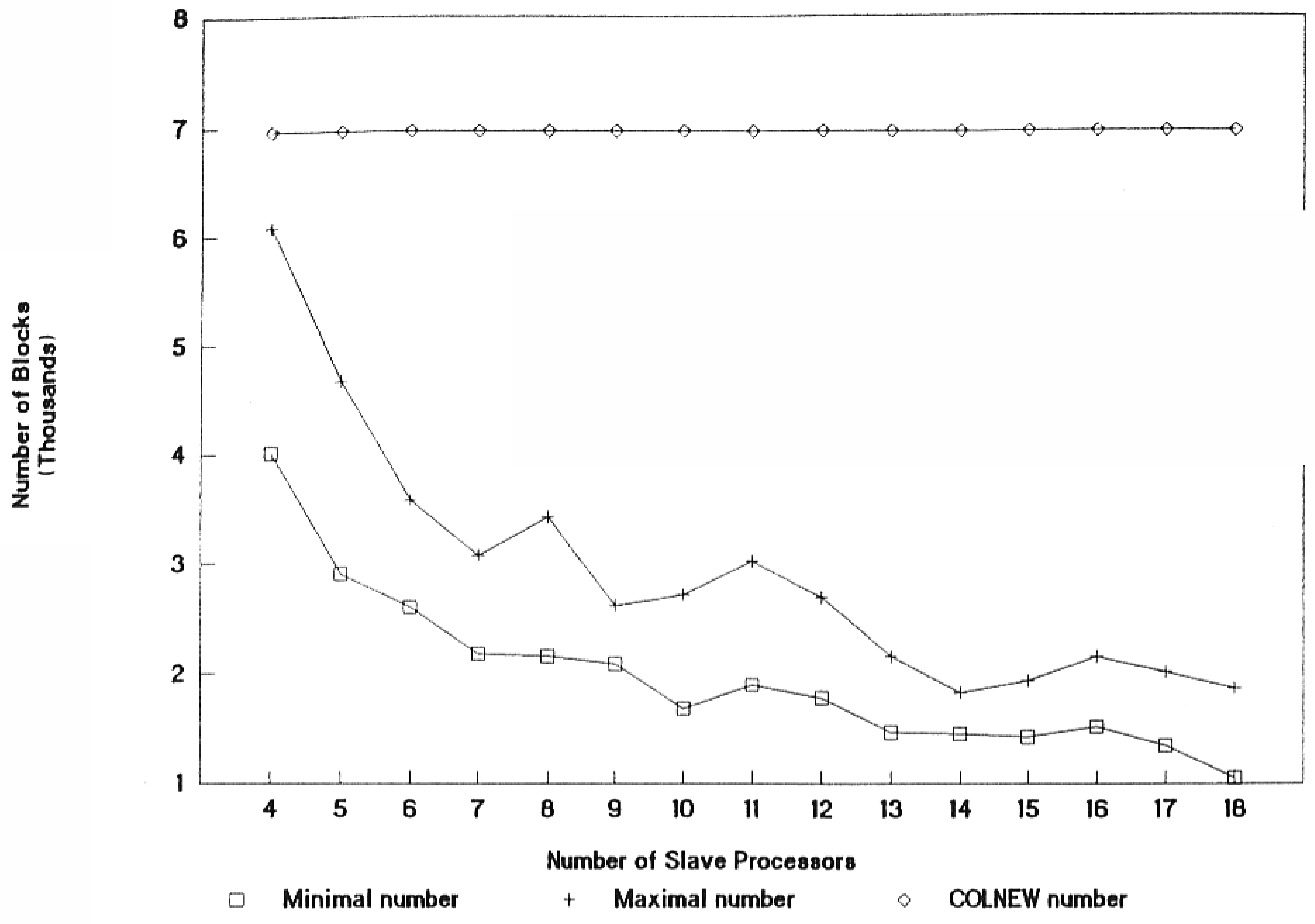

Fig. 9. Blocks used per processor (Problem 28).

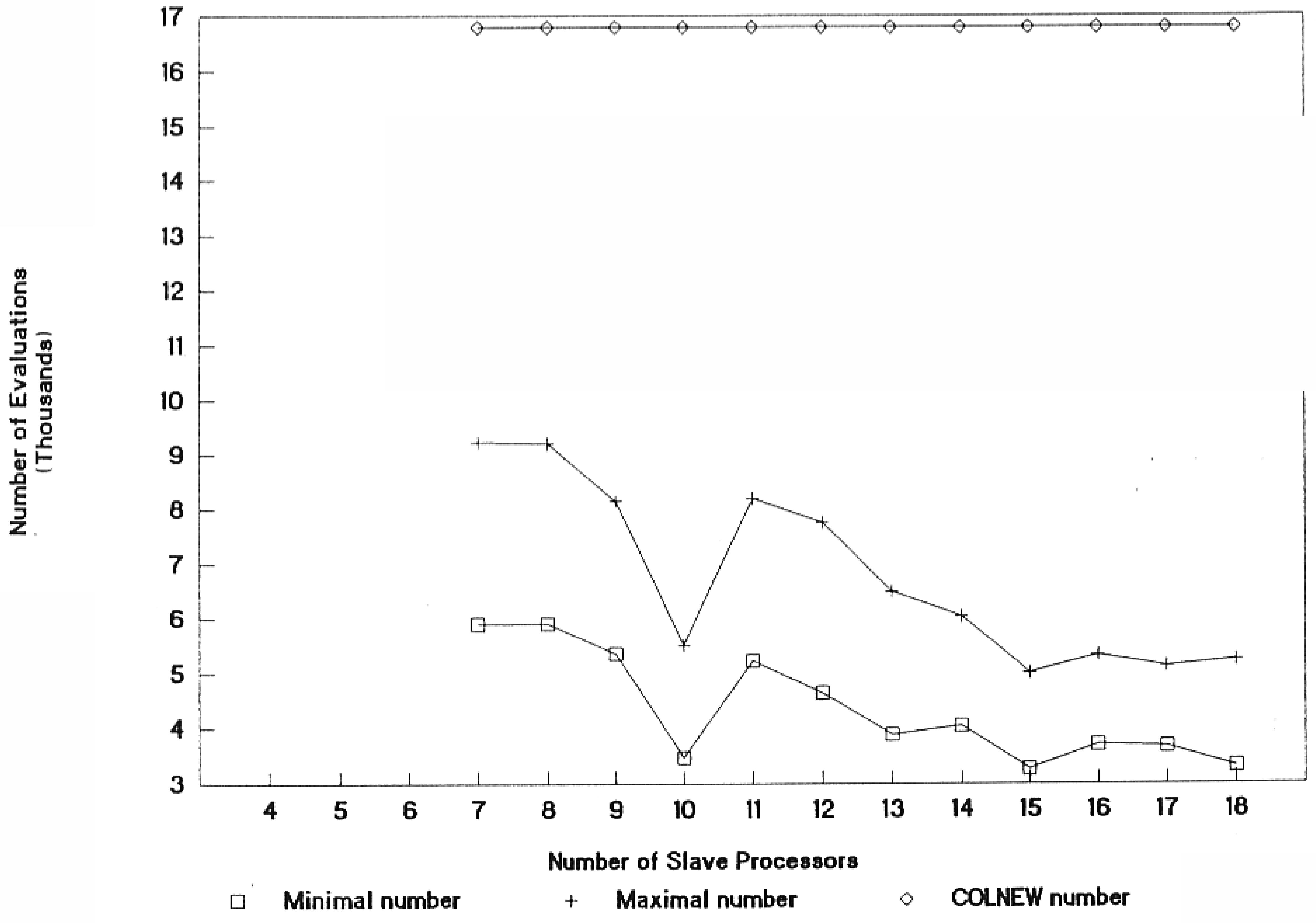

Fig. 10. Function evaluations per processor (Problem 29).

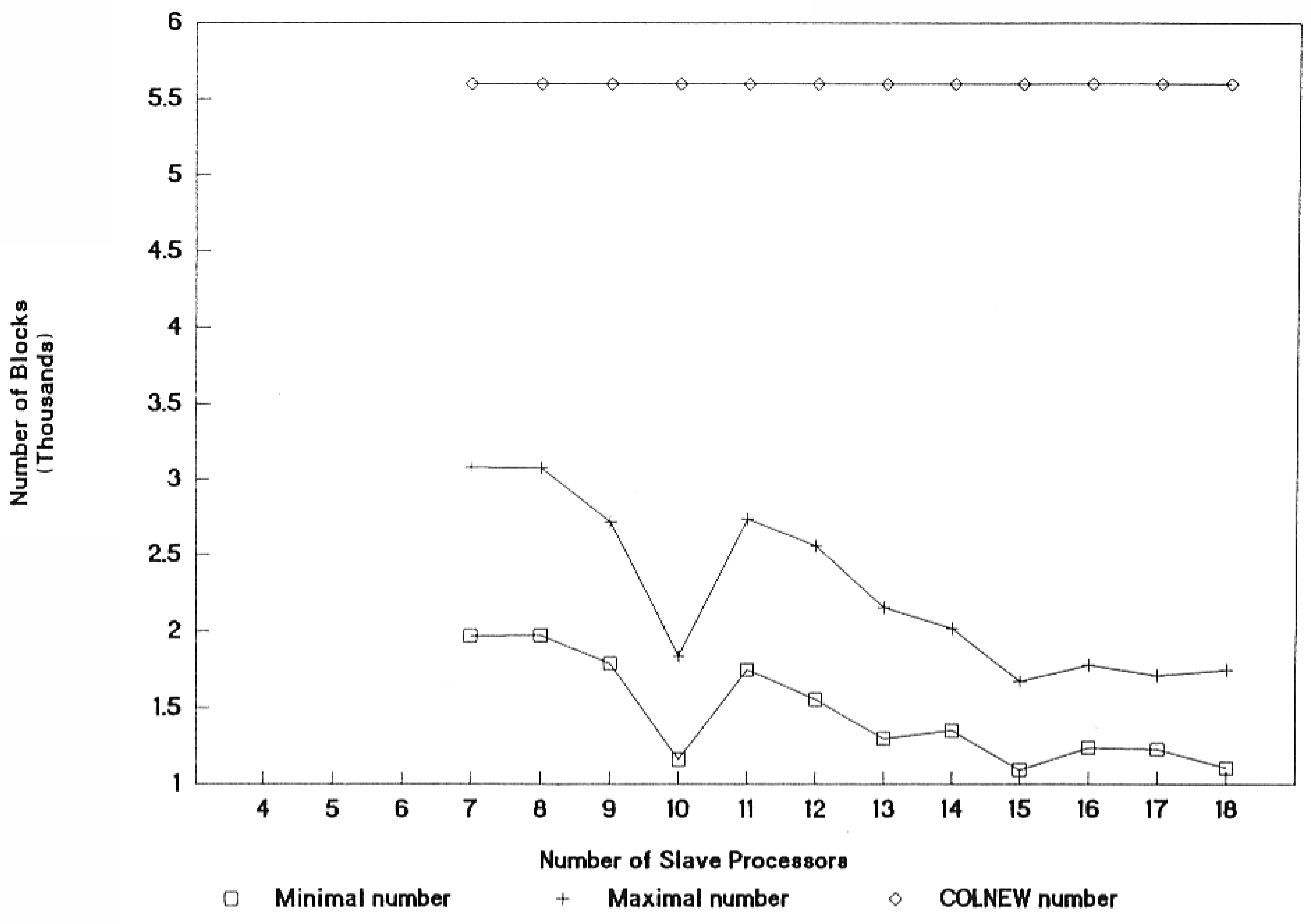

Fig. 11. Blocks used per processor (Problem 29).

The large values of Mxd and of Stp indicate that the program often has difficulties making the decision to chop and instead doubles the mesh. Each doubling increases the size of the linear system to be solved. For instance when $\mathrm{Mxd} = 3$ and $P = 5$, the linear systems solved have approximately 100 blocks. This effect is especially clear for (25). For all P, the value of $\mathrm{Nb}_{\max}$ is larger than Nb. Also, since $\mathrm{Mxd} > 0$, the decision not to increase the number of main meshpoints is a clear handicap. However this did not prevent significant speedups for (26) and (27).

The quality of the solution at accepted mesh points is perhaps surprisingly high. In complete agreement with the discussion in Section 4 there was no significant growth of error in the solution resulting from chopping. In each case the number of correct digits (Ncd) was calculated by comparing the calculated solution on the final mesh with the analytic solution. The results represent the minimum Ncd in the whole interval on the final accepted mesh and solution. The Ncds for COLNEW and for the new program are similar (except for (27) where the accuracy achieved by COLNEW was very high). The overall error in the final solution was smoother for COLNEW.

The imbalance between the work on the 'busiest' and the 'laziest' processor measured in terms of number of the function evaluations performed (or the number of blocks used in the linear algebra) is up to 40%. In addition there is erratic behavior of $\mathrm{Nf}_{\max}$ (and $\mathrm{Nb}_{\max}$) for increasing numbers of processors, P. These suggest scheduling problems typical of large granularity parallelization.

6. Conclusions

The parallel chopping algorithm is based on a set of specially designed meshes. Experiments show that this approach has the usual disadvantages of macroscale parallelization. It also suffers from the crudeness of the chopping procedure. The methods described in [12] and [13] may be used (independently or combined) to improve the performance of the BVP solver. They can also be used to divide work in a more flexible way between processors. A more sophisticated tool than FORCE is needed to implement these embedded algorithms.

Despite the deficiencies in algorithmic design and the overhead of parallel computation, quite impressive speedups are achieved for two of three problems when using a large enough number of processors. It seems clear that some of the speedup is due to deficiencies in COLNEW's strategy for mesh redistribution and to a poor initial mesh choice. But the parallel program could also be speeded up by an improved main meshpoint strategy, by performing single and double mesh solutions in parallel and, possibly, by treating all active subintervals simultaneously. Somewhat surprisingly it is superior when there are only one or two 'difficult' regions in the solution. This suggests that there is scope for an improved mesh selection strategy in COLNEW. Alternatively, in a parallel environment, the procedure described here may provide a better mesh selection strategy for COLNEW. Mesh selection has been identified in [5] as one of the major remaining bottlenecks in the parallel solution of boundary value problems.

Acknowledgement

We are grateful for the comments and constructive suggestions of Professors L.F. Shampine and W.E. Ferguson. Ian Gladwell wishes to acknowledge the support of NATO Grant CRA 920037.

References

[1] U. Ascher, J. Christiansen and R.D. Russell, Collocation software for boundary value ODEs, *ACM Trans. Math. Soft.* 7 (1981) 209–229.

[2] U. Ascher and S.Y.P. Chan, On parallel methods for boundary value ODE's, University of British Columbia, Department of Computer Science, Technical Report 89-19, 1989.

[3] U. Ascher, R.M.M. Mattheij and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1988).

[4] A. Bader and U. Ascher, A new basis implementation for a mixed-order boundary value ODE solver, *SIAM J. Sci. Stat. Comp.* 8 (1987) 483–500.

[5] K. Bennett and G. Fairweather, PCOLNEW: A parallel boundary-value solver for shared memory machines, U. of Kentucky, Tech. Rept. CCS-90-8, Center for Comp. Sc., 1990.

[6] J. Dongarra, J.J. Du Croz, I. Duff and S. Hammarling, A set of level 3 basic linear algebra subprograms, *ACM Trans. Math. Soft.* 16 (1990) 1–17.

[7] I. Gladwell and R.I. Hay, Vector- and parallelization of ODE BVP codes, *Parallel Comput.* 12 (1989) 343–350.

[8] R.I. Hay, The impact of vector processors on boundary value codes, Ph.D. Thesis, University of Manchester, Department of Mathematics, England, 1986.

[9] P.W. Hemker, A numerical study of stiff two-point boundary problems, Mathematisch Centrum, Amsterdam, 1977.

[10] H.F. Jordan, M.S. Benten and N.S. Arenstorf, Force user's manual, University of Colorado, Boulder, Colorado, 1986.

[11] M. Paprzycki, Parallelization of boundary value problem software, Ph.D. Thesis, Department of Mathematics, Southern Methodist University, Dallas, 1990.

[12] M. Paprzycki and I.Gladwell, Solving almost block diagonal systems on parallel computers, *Parallel Comput.* 17 (1991) 133–153.

[13] M. Paprzycki and I. Gladwell, Solving almost block diagonal systems using level 3 BLAS, in: J. Dongerra et al., eds., *Proc. 5th SIAM Conf. on Parallel Processing in Scientific Computation*, (SIAM, Philadelphia, PA, 1992) 52–62.

[14] R.D. Russell and J. Christiansen, Adaptive mesh selection strategies for solving BVP's, *SIAM J. Numer. Anal.* 15 (1978) 59–80.

[15] S.J. Wright and V. Pereyra, Adaptation of a two-point boundary value solver to a vector-multiprocessor environment, *SIAM J. Sci. Stat. Comp.* 11 (1990) 425–449.