Parallel implementation of a monotone domain decomposition algorithm for nonlinear reaction-diffusion problems

M. P. Hardy * I. Boglaev†

(received 21 October 2004, revised 23 March 2005)

Abstract

Recently, a monotone iterative domain decomposition algorithm has been proposed for the numerical solution of nonlinear singularly perturbed reaction-diffusion problems. This paper describes a parallel implementation of the algorithm on a distributed memory cluster. Interprocess communication is effected by means of the MPI message passing library. For various domain decompositions, we give the convergence iteration count and execution time on up to 16 processors. The parallel scale-up of the algorithm improves as the number of mesh points is increased.

*Mathematical Sciences Institute, Australian National University, Canberra, Australia. mailto:hardy@maths.anu.edu.au

†Institute of Fundamental Sciences, Massey University, Palmerston North, New Zealand mailto:i.boglaev@massey.ac.nz

See http://anziamj.austms.org.au/V46/CTAC2004/Hard for this article, Published April 28, 2005. ISSN 1446-8735

Contents

1 Introduction

We are interested in the nonlinear singularly perturbed reaction-diffusion problem of elliptic type

$$\begin{aligned} -\mu^2(u_{xx}+u_{yy})+f(x,y,u)=0\,, \quad (x,y)\in\omega\,,\\ \omega=\omega^x\times\omega^y=(0,1)\times(0,1)\,, \quad u=g \text{ on } \partial\omega\,, \end{aligned} \tag{1}$$

where $\mu \ll 1$ is the perturbation parameter and $\partial\omega$ is the boundary of ω. We also assume that $c^* \geq f_u \geq c_*$, $(x,y,u)\in\bar{\omega}\times\mathbb{R}$, where c^* and c_* are positive constants and $f_u \equiv \partial f/\partial u$. The solution is characterised by boundary layers of width $\mathcal{O}(\mu|\ln\mu|)$.

Discrete approximation of (1) leads to an algebraic system of nonlinear difference equations whose solution converges with mesh refinement to that of the continuous problem. The algebraic system is typically solved by Newton's method, or some other iterative technique. One drawback of Newton's method is its sensitivity to the initial guess. In contrast, the method of upper and lower solutions generates a monotonically convergent sequence from any

one of a wide class of initial iterates [4]. Indeed, as shown in [1], the initial iterate may be constructed using only the boundary conditions and a lower bound on f_u. No knowledge of the solution is necessary to implement the algorithm.

The advent of the Beowulf cluster has brought high-performance computing within reach of academe and thus fostered renewed interest in alternating Schwarz-type domain decomposition iterative algorithms. In [2] the domain is partitioned into nonoverlapping boxes and the monotone iterative method is applied on each subdomain. At each horizontal and vertical boundary, interfacial subdomains are introduced and corresponding linear problems generate boundary Dirichlet data for the nonoverlapping subdomains. As shown theoretically and confirmed by serial computations [2], the algorithm retains global monotonicity under such decomposition. This paper describes a parallel implementation of the algorithm from [2].

In Section 2 we define the piecewise uniform mesh from [3], on which the classical central difference scheme converges μ-uniformly to the solution of (1). We then describe the domain decomposition algorithm from [2]. In Section 3 we discuss our parallel implementation of the algorithm. Section 4 presents numerical experiments for a model problem. For various domain decompositions we give the convergence iteration counts and execution times on up to 16 processors. Observing that domain decomposition enhances the algorithm's serial execution, we define parallel speedup in terms of the optimal decomposition for a given number of processors. The parallel scale-up of the algorithm improves as the mesh size is increased.

2 Box-domain decomposition algorithm

We provide here only a brief description of the nonlinear finite difference scheme for (1) and the monotone domain decomposition algorithm by which it is solved. Further details can be found in [2].

Let N_x and N_y be the number of mesh subintervals in the respective x- and y- directions. Boundary layer thicknesses σ_x and σ_y are chosen as

$$\sigma_x = \min\left\{0.25, \mu c_*^{-1/2} \ln N_x\right\}, \qquad \sigma_y = \min\left\{0.25, \mu c_*^{-1/2} \ln N_y\right\},$$

and mesh spacings

$$h_{x\mu} = \frac{4\sigma_x}{N_x}, \quad h_x = \frac{2(1-2\sigma_x)}{N_x}, \quad h_{y\mu} = \frac{4\sigma_y}{N_y}, \quad h_y = \frac{2(1-2\sigma_y)}{N_y}.$$

The x-mesh $\bar{\omega}^{hx}$ is constructed thus: in each of the subintervals $[0, \sigma_x]$ and $[1-\sigma_x, 1]$ the fine mesh spacing is $h_{x\mu}$ whereas in the interval $[\sigma_x, 1-\sigma_x]$ the coarse mesh spacing is h_x. The y-mesh $\bar{\omega}^{hy}$ is constructed analogously. The mesh for $\bar{\omega}$ is then defined as the tensor product $\bar{\omega}^h = \bar{\omega}^{hx} \times \bar{\omega}^{hy}$.

Given a mesh function U on $\bar{\omega}^h$, let $\mathcal{D}_x^2 U(P)$ and $\mathcal{D}_y^2 U(P)$ be the respective central difference approximations to the x- and y- second derivatives at point $P \in \omega^h$. Define also the linear operator $\mathcal{L}^h \equiv -\mu^2(\mathcal{D}_x^2 + \mathcal{D}_y^2)$. For discrete approximation of (1) we use the classical central difference scheme

$$\mathcal{L}^h U + f(P, U) = 0, \quad P \in \omega^h, \quad U = g \text{ on } \partial\omega^h. \tag{2}$$

On the mesh $\bar{\omega}^h$ this scheme converges μ-uniformly to the solution u of (1). That is, there exists a constant C, independent of μ and N, such that

$$\max_{P \in \bar{\omega}^h} |U(P) - u(P)| \le C\left(N^{-1} \ln N\right)^2, \quad N = \min\{N_x, N_y\}.$$

To solve the nonlinear difference scheme (2), we employ the box-domain decomposition algorithm from [2]. Consider a decomposition of ω into $M \times L$ nonoverlapping main subdomains $\omega_{m,l}$, $m = 1, \ldots, M$, $l = 1, \ldots, L$:

$$\omega_{m,l} = (x_{m-1}, x_m) \times (y_{l-1}, y_l), \quad x_0 = 0, \quad x_M = 1, \quad y_0 = 0, \quad y_L = 1.$$

Then introduce vertical interfacial subdomains θ_m, $m = 1, \ldots, M-1$:

$$\theta_m = (x_m^b, x_m^e) \times \omega^y, \quad x_m^b < x_m < x_m^e, \quad \theta_{m-1} \cap \theta_m = \emptyset,$$

and horizontal interfacial subdomains ϑ_l, $l = 1, \ldots, L-1$:

$$\vartheta_l = \omega^x \times (y_l^b, y_l^e)\,, \quad y_l^b < y_l < y_l^e\,, \quad \vartheta_{l-1} \cap \vartheta_l = \emptyset\,.$$

With the global mesh $\bar{\omega}^h = \bar{\omega}^{hx} \times \bar{\omega}^{hy}$, we also require that

$$\{x_m^{b,e}, x_m\} \subset \omega^{hx}\,,\ m = 1, \ldots, M-1\,, \quad \{y_l^{b,e}, y_l\} \subset \omega^{hy}\,,\ l = 1, \ldots, L-1\,.$$

1. On the whole mesh $\bar{\omega}^h$ choose an initial mesh function $V^{(0)}$ satisfying the boundary conditions, $V^{(0)}(P) = g(P)$, $P \in \partial\omega^h$.
   Given a global iterate $V^{(n)}$, Steps 2 through 5 below generate $V^{(n+1)}$.

2. For each main subdomain $\bar{\omega}_{m,l}^h$, solve the linear difference problem
   $$(\mathcal{L}^h + c^*) Z_{m,l}^{(n+1)} = -\left[\mathcal{L}^h V^{(n)} + f(P, V^{(n)})\right]\,, \quad P \in \omega_{m,l}^h = \omega_{m,l} \cap \omega^h\,,$$
   with $Z_{m,l}^{(n+1)}(\partial\omega_{m,l}^h) = 0$.

3. For each vertical interfacial subdomain solve the linear problem
   $$(\mathcal{L}^h + c^*) Z_m^{(n+1)} = -\left[\mathcal{L}^h V^{(n)} + f(P, V^{(n)})\right]\,, \quad P \in \theta_m^h = \theta_m \cap \omega^h\,,$$
   with $Z_m^{(n+1)}(\partial\theta_m^h)$ defined by the mesh functions computed in Step 2.

4. For each horizontal interfacial subdomain solve the linear problem
   $$(\mathcal{L}^h + c^*) \tilde{Z}_l^{(n+1)} = -\left[\mathcal{L}^h V^{(n)} + f(P, V^{(n)})\right], \quad P \in \vartheta_l^h = \vartheta_l \cap \omega^h\,,$$
   with $\tilde{Z}_l^{(n+1)}(\partial\vartheta_l^h)$ defined by the mesh functions computed in Steps 2 and 3.

5. Piece together the mesh functions from Steps 2 through 4:
   $$V^{(n+1)}(P) = \begin{cases} V^{(n)}(P) + \tilde{Z}_l^{(n+1)}(P)\,, & P \in \bar{\vartheta}_l^h\,, \\ V^{(n)}(P) + Z_m^{(n+1)}(P), & P \in \bar{\theta}_m^h \setminus \left\{\bigcup_{l=1}^{L-1} \bar{\vartheta}_l^h\right\}, \\ V^{(n)}(P) + Z_{m,l}^{(n+1)}(P)\,, & P \in \bar{\omega}_{m,l}^h \setminus \left\{\bigcup_{m=1}^{M-1} \bar{\theta}_m^h \bigcup_{l=1}^{L-1} \bar{\vartheta}_l^h\right\}. \end{cases}$$

6. If the solution is not converged then increment n and go to Step 2.

3 Parallel implementation

We have implemented the box-domain decomposition algorithm on *Helix*—the distributed memory Linux cluster at Massey University, New Zealand. This comprises 65 nodes, each equipped with two Athlon MP-2100 processors, 1 GB of RAM and two gigabit network interface cards. The nodes are arranged on seven 24 port switches such that any two nodes are connected via at most three switches. Inter-processor communication is via the MPI library specification.

Because the mesh is only piecewise uniform, the linear systems may be nonsymmetric. Therefore, we solve all linear systems with the restarted GMRES(m) algorithm from [5] and the point Jacobi preconditioner. The present work concerns the first level of parallelisation. That is, each subdomain is wholly assigned to a processor and we do not parallelise GMRES. Hence, we only consider balanced domain decompositions — those in which the interfacial subdomains overlap the boundary layer. Although the analysis and serial computations of [2] have shown that unbalanced domain decompositions require fewer iterations for convergence, this advantage would be at least partially offset by the extra inter-processor communication required.

In a balanced domain decomposition, the main subdomains share the mesh points equally. However, the respective workloads of the corresponding linear problems vary with mesh spacing. In order to balance the load of Step 2, we first classify the main subdomains by mesh spacing, and then divide each class equally among the processors. Suppose we have an $M \times L$ decomposition and P processors. Assuming that $M \geq 4$ and $L \geq 4$, there are $ML/16$ main subdomains in each of the four corners $[0, \sigma_x] \times [0, \sigma_y]$, $[1-\sigma_x, 1] \times [0, \sigma_y]$, $[0, \sigma_x] \times [1-\sigma_y, 1]$ and $[1-\sigma_x, 1] \times [1-\sigma_y, 1]$. Thus, there are $ML/4$ main subdomains in which the respective x- and y- mesh spacings are $h_{x\mu}$ and $h_{y\mu}$. Let us denote this class by F-F (fine-fine). Next, in each of the regions $[\sigma_x, 1-\sigma_x] \times [0, \sigma_y]$, $[\sigma_x, 1-\sigma_x] \times [1-\sigma_y, 1]$, $[0, \sigma_x] \times [\sigma_y, 1-\sigma_y]$ and $[1-\sigma_x, 1] \times [\sigma_y, 1-\sigma_y]$, there are $ML/8$ main subdomains in which the

respective x- and y- mesh spacings are either $h_{x\mu}$ and h_y or h_x and $h_{y\mu}$. In the numerical experiments of the next section we take $N_x = N_y$ and hence these $ML/2$ subdomains are equivalent. We label this class F-C (fine-coarse). Finally, the region $[\sigma_x, 1-\sigma_x] \times [\sigma_y, 1-\sigma_y]$ is covered by $ML/4$ main subdomains in which the respective x- and y- mesh spacings are h_x and h_y. We denote this class C-C (coarse-coarse). Within a given class, each subdomain incurs the same computational cost. Comparing across classes, the cost per subdomain is greatest for the class F-F and least for the class C-C. If P divides $ML/4$, each processor is assigned $ML/(4P)$ main subdomains of class F-F, $ML/(2P)$ main subdomains of class F-C and $ML/(4P)$ main subdomains of class C-C. If P does not divide $ML/4$, then load balancing requires second level parallelisation and we do not implement the $M \times L$ decomposition on P processors. Similar considerations apply to decompositions in which M or L is equal to one, except that there are at most two classes of main subdomain. The vertical interfacial subdomains are shared as equally as possible among the P processors $\{0, 1, \ldots, P-1\}$: $\bar{\theta}^h_m \mapsto \mod(m-1, P)$. The assignment of horizontal interfacial subdomains is similar: $\bar{\vartheta}^h_l \mapsto \mod(l-1, P)$.

Consider the distribution of an 8×4 decomposition on four processors, shown schematically in Figure 1. During Step 2 of the algorithm, each processor solves over its assigned main subdomains and this workload is balanced. During Step 3, each processor solves over its assigned vertical interfacial subdomains. In contrast to the other processors which each have two, Processor 3 has only one vertical interfacial subdomain and so is idle for approximately half of Step 3. In Step 4, Processors 0,1,2 each solve over their assigned horizontal interfacial subdomain while Processor 3 remains idle. This idle time results in a loss of computational efficiency.

Another overhead of any parallel implementation is that of inter-processor communication. Before each of the algorithm's three steps, data must be transferred between subdomains. In Figure 1 we have indicated the data which must be transferred from Processor 0 to Processor 1 before the latter

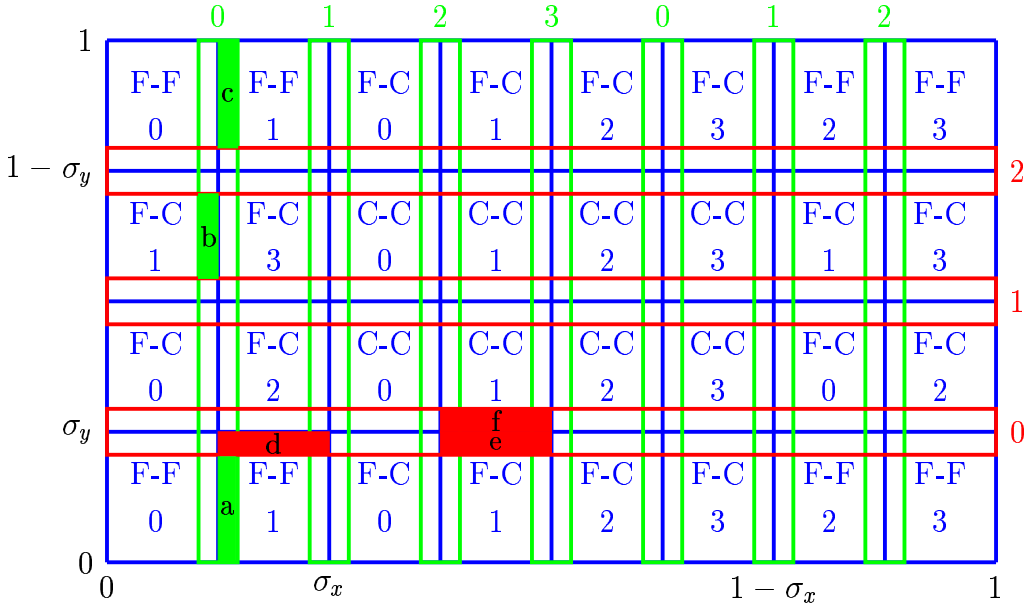

FIGURE 1: A schematic of an 8×4 decomposition and the assignment of the subdomains to four processors $\{0, 1, 2, 3\}$. Main subdomains are shown in blue, vertical interfacial subdomains are shown in green and horizontal interfacial subdomains are shown in red. The mesh spacing classes of the main subdomains are also indicated. The six blocks of data `a`–`f` must be transferred from Processor 0 to Processor 1 before Step 2 of the algorithm.

can solve over its main subdomains. From the left-most vertical interfacial subdomain on Processor 0, we must transfer the three blocks of data `a`, `b` and `c` to main subdomains on Processor 1. From the horizontal interfacial subdomain on Processor 0, we must transfer the three blocks of data `d`, `e` and `f` to Processor 1. In order to minimise the inter-processor communication, the data blocks `a`–`f` are buffered and sent as one message. This saving in communication comes at the relatively small (local) cost of composing the message on Processor 0 and decomposing the received message on Processor 1.

4 Numerical experiments

We now apply the algorithm to the reaction-diffusion problem

$$-\mu^2(u_{xx}+u_{yy})+\frac{u-4}{5-u}=0\,,\quad (x,y)\in\omega\,,$$
$$\omega=\omega^x\times\omega^y=(0,1)\times(0,1)\,,\quad u(x,y)=1\,,\quad (x,y)\in\partial\omega\,.$$

The solution to the reduced problem ($\mu=0$) is $u_r=4$. For $\mu\ll 1$ the problem is singularly perturbed and the solution increases sharply from $u=1$ on $\partial\omega$ to $u=4$ on the interior.

We solve the nonlinear scheme (2) by the domain decomposition algorithm of Section 2. We fix the perturbation parameter $\mu=10^{-3}$ and take $N_x=N_y=N$ with $N=256$, 512 or 1024. Given P processors, where $P\in\{1,2,4,8,16\}$, we are interested in the execution time of the algorithm under various domain decompositions. We suppose that $\{M,L\}\subset\{1,4,8,16,32\}$ and that the interfacial subdomains are either all maximal or all minimal. We initiate the algorithm with the mesh function $V^{(0)}(\omega^h)=0$, $V^{(0)}(\partial\omega^h)=1$. As shown in [2], this guarantees monotonic convergence to the solution of (2). Our convergence criterion is $\|V^{(n)}-V^{(n-1)}\|_{\bar\omega^h}\le 10^{-5}$.

N	256					512					1024				
$L\backslash M$	1	4	8	16	32	1	4	8	16	32	1	4	8	16	32
1	21	$\frac{26}{21}$	$\frac{40}{21}$	$\frac{41}{22}$	$\frac{49}{24}$	21	$\frac{36}{21}$	$\frac{62}{21}$	$\frac{62}{21}$	$\frac{72}{23}$	21	$\frac{59}{21}$	$\frac{102}{21}$	$\frac{102}{21}$	$\frac{113}{23}$
4	$\frac{26}{21}$	$\frac{26}{21}$	$\frac{40}{21}$	$\frac{41}{22}$	$\frac{49}{24}$	$\frac{36}{21}$	$\frac{36}{21}$	$\frac{62}{21}$	$\frac{62}{21}$	$\frac{72}{23}$	$\frac{59}{21}$	$\frac{59}{21}$	$\frac{102}{21}$	$\frac{102}{21}$	$\frac{113}{23}$
8	$\frac{40}{21}$	$\frac{40}{21}$	$\frac{41}{21}$	$\frac{43}{22}$	$\frac{55}{24}$	$\frac{62}{21}$	$\frac{62}{21}$	$\frac{72}{21}$	$\frac{72}{21}$	$\frac{86}{23}$	$\frac{102}{21}$	$\frac{102}{21}$	$\frac{126}{21}$	$\frac{126}{21}$	$\frac{143}{23}$
16	$\frac{41}{22}$	$\frac{41}{22}$	$\frac{43}{22}$	$\frac{44}{22}$	$\frac{57}{24}$	$\frac{62}{21}$	$\frac{62}{21}$	$\frac{72}{21}$	$\frac{73}{21}$	$\frac{88}{23}$	$\frac{102}{21}$	$\frac{102}{21}$	$\frac{126}{21}$	$\frac{127}{21}$	$\frac{144}{22}$
32	$\frac{49}{24}$	$\frac{49}{24}$	$\frac{55}{24}$	$\frac{57}{24}$	$\frac{67}{24}$	$\frac{72}{23}$	$\frac{72}{23}$	$\frac{86}{23}$	$\frac{88}{23}$	$\frac{103}{23}$	$\frac{113}{23}$	$\frac{113}{23}$	$\frac{143}{23}$	$\frac{144}{22}$	$\frac{163}{23}$

TABLE 1: The convergence iteration count of the algorithm for various $M \times L$ decompositions. Results corresponding to minimal and maximal interfacial subdomains are given above and below the line, respectively.

Table 1 shows convergence iteration counts for various mesh sizes N and $M \times L$ decompositions. For each N, the iteration count of the undecomposed algorithm ($M = 1\,,\ L = 1$) is 21 and this increases under decomposition. The convergence improves as the interfacial subdomains are enlarged.

In Table 2 we list the execution time of the $M \times L$ algorithm on P processors. We first discuss the results for $P = 1$ (the serial code) and $N = 256\,,$ corresponding to the major cell in the top-left corner of Table 2. The undecomposed algorithm executes in 34 seconds. Each of the 21 iterations entails the solution of a linear problem with 255^2 unknowns. Consider now the 4×4 decomposition with minimal interfacial subdomains. Each global iteration entails the solution of 22 linear problems, 16 of which have 63^2 unknowns (Step 2 of the algorithm) and 6 of which have 255 unknowns (Steps 3 and 4). Now, the GMRES operation count increases superlinearly with problem size. Therefore, the above-mentioned 22 linear problems are solved more quickly than the undecomposed algorithm's linear problem. Indeed, the 26 global iterations execute in just 15 seconds.

We now discuss the communication overhead of our implementation. Consider the problem with $N = 256$ and the 32×32, minimal interfacial sub-

	N	256					512					1024				
P	$L\backslash M$	1	4	8	16	32	1	4	8	16	32	1	4	8	16	32
1	1	34	$\frac{32}{46}$	$\frac{29}{33}$	$\frac{25}{28}$	$\frac{15}{17}$	249	$\frac{356}{377}$	$\frac{497}{394}$	$\frac{400}{333}$	$\frac{252}{215}$	2037	$\frac{4014}{3116}$	$\frac{5665}{3281}$	$\frac{5394}{3231}$	$\frac{4798}{2956}$
	4	$\frac{32}{46}$	$\frac{15}{51}$	$\frac{14}{42}$	$\frac{7}{38}$	$\frac{7}{34}$	$\frac{356}{376}$	$\frac{233}{458}$	$\frac{239}{430}$	$\frac{196}{393}$	$\frac{120}{324}$	$\frac{4014}{3109}$	$\frac{3069}{3964}$	$\frac{4895}{4059}$	$\frac{4174}{3826}$	$\frac{2484}{3481}$
	8	$\frac{29}{33}$	$\frac{14}{42}$	$\frac{9}{34}$	$\frac{8}{33}$	$\frac{9}{28}$	$\frac{497}{394}$	$\frac{240}{432}$	$\frac{272}{434}$	$\frac{179}{387}$	$\frac{99}{324}$	$\frac{5642}{3272}$	$\frac{4856}{4058}$	$\frac{5482}{3944}$	$\frac{3674}{3565}$	$\frac{3188}{3493}$
	16	$\frac{24}{28}$	$\frac{7}{38}$	$\frac{8}{32}$	$\frac{7}{30}$	$\frac{8}{26}$	$\frac{400}{332}$	$\frac{196}{393}$	$\frac{179}{385}$	$\frac{91}{343}$	$\frac{91}{299}$	$\frac{5348}{3245}$	$\frac{4148}{3849}$	$\frac{3674}{3580}$	$\frac{3330}{3439}$	$\frac{2255}{3193}$
	32	$\frac{15}{17}$	$\frac{7}{35}$	$\frac{8}{28}$	$\frac{8}{26}$	$\frac{7}{18}$	$\frac{251}{212}$	$\frac{119}{326}$	$\frac{97}{325}$	$\frac{90}{300}$	$\frac{83}{223}$	$\frac{4789}{2960}$	$\frac{2469}{3490}$	$\frac{3182}{3509}$	$\frac{2254}{3218}$	$\frac{1144}{2949}$
2	1	$*$	$\frac{22}{34}$	$\frac{17}{20}$	$\frac{14}{16}$	$\frac{9}{10}$	$*$	$\frac{220}{268}$	$\frac{307}{259}$	$\frac{237}{207}$	$\frac{137}{117}$	$*$	$\frac{2553}{2241}$	$\frac{3597}{2137}$	$\frac{3388}{2043}$	$\frac{3029}{1876}$
	4	$\frac{22}{35}$	$\frac{9}{37}$	$\frac{8}{29}$	$\frac{4}{26}$	$\frac{5}{25}$	$\frac{220}{268}$	$\frac{139}{346}$	$\frac{129}{304}$	$\frac{102}{272}$	$\frac{63}{227}$	$\frac{2554}{2234}$	$\frac{1951}{3037}$	$\frac{3091}{2893}$	$\frac{2516}{2682}$	$\frac{1354}{2440}$
	8	$\frac{17}{20}$	$\frac{8}{29}$	$\frac{5}{21}$	$\frac{5}{20}$	$\frac{6}{18}$	$\frac{307}{259}$	$\frac{129}{304}$	$\frac{141}{286}$	$\frac{94}{251}$	$\frac{52}{205}$	$\frac{3603}{2118}$	$\frac{3092}{2902}$	$\frac{3324}{2619}$	$\frac{2003}{2302}$	$\frac{1659}{2233}$
	16	$\frac{14}{16}$	$\frac{4}{26}$	$\frac{5}{20}$	$\frac{5}{18}$	$\frac{6}{16}$	$\frac{234}{206}$	$\frac{102}{271}$	$\frac{93}{250}$	$\frac{48}{216}$	$\frac{48}{180}$	$\frac{3384}{2042}$	$\frac{2514}{2689}$	$\frac{1991}{2320}$	$\frac{1729}{2177}$	$\frac{1181}{2023}$
	32	$\frac{8}{10}$	$\frac{4}{25}$	$\frac{6}{18}$	$\frac{6}{16}$	$\frac{8}{14}$	$\frac{135}{118}$	$\frac{64}{227}$	$\frac{51}{206}$	$\frac{48}{183}$	$\frac{46}{128}$	$\frac{3026}{1881}$	$\frac{1340}{2434}$	$\frac{1663}{2243}$	$\frac{1177}{2040}$	$\frac{593}{1869}$
4	1	$*$	$*$	$\frac{9}{11}$	$\frac{7}{9}$	$\frac{5}{5}$	$*$	$*$	$\frac{187}{148}$	$\frac{141}{116}$	$\frac{76}{64}$	$*$	$*$	$\frac{1965}{1176}$	$\frac{1797}{1066}$	$\frac{1580}{962}$
	4	$*$	$\frac{5}{22}$	$\frac{5}{17}$	$\frac{3}{16}$	$\frac{3}{15}$	$*$	$\frac{80}{186}$	$\frac{71}{169}$	$\frac{55}{145}$	$\frac{34}{121}$	$*$	$\frac{988}{1621}$	$\frac{1570}{1605}$	$\frac{1293}{1416}$	$\frac{698}{1293}$
	8	$\frac{9}{11}$	$\frac{5}{17}$	$\frac{3}{13}$	$\frac{3}{12}$	$\frac{3}{11}$	$\frac{187}{148}$	$\frac{71}{169}$	$\frac{75}{164}$	$\frac{49}{138}$	$\frac{27}{117}$	$\frac{1959}{1175}$	$\frac{1565}{1591}$	$\frac{1688}{1495}$	$\frac{1017}{1278}$	$\frac{847}{1257}$
	16	$\frac{8}{9}$	$\frac{3}{15}$	$\frac{3}{12}$	$\frac{3}{10}$	$\frac{4}{9}$	$\frac{141}{114}$	$\frac{58}{146}$	$\frac{51}{139}$	$\frac{26}{114}$	$\frac{26}{95}$	$\frac{1790}{1059}$	$\frac{1289}{1417}$	$\frac{1011}{1280}$	$\frac{879}{1147}$	$\frac{598}{1058}$
	32	$\frac{5}{5}$	$\frac{3}{15}$	$\frac{3}{11}$	$\frac{4}{9}$	$\frac{5}{8}$	$\frac{76}{62}$	$\frac{34}{122}$	$\frac{26}{118}$	$\frac{25}{95}$	$\frac{24}{66}$	$\frac{1578}{956}$	$\frac{713}{1293}$	$\frac{843}{1251}$	$\frac{597}{1062}$	$\frac{305}{967}$
8	1	$*$	$*$	$*$	$\frac{4}{5}$	$\frac{3}{3}$	$*$	$*$	$*$	$\frac{77}{61}$	$\frac{42}{33}$	$*$	$*$	$*$	$\frac{1016}{542}$	$\frac{874}{478}$
	4	$*$	$*$	$\frac{3}{14}$	$\frac{2}{13}$	$\frac{2}{13}$	$*$	$*$	$\frac{36}{127}$	$\frac{31}{112}$	$\frac{19}{102}$	$*$	$*$	$\frac{779}{1140}$	$\frac{658}{1020}$	$\frac{371}{970}$
	8	$*$	$\frac{3}{14}$	$\frac{2}{8}$	$\frac{2}{7}$	$\frac{2}{6}$	$*$	$\frac{37}{127}$	$\frac{38}{93}$	$\frac{26}{77}$	$\frac{15}{64}$	$*$	$\frac{781}{1131}$	$\frac{847}{833}$	$\frac{524}{693}$	$\frac{431}{658}$
	16	$\frac{4}{5}$	$\frac{2}{13}$	$\frac{2}{7}$	$\frac{2}{6}$	$\frac{3}{5}$	$\frac{76}{60}$	$\frac{31}{112}$	$\frac{27}{77}$	$\frac{14}{61}$	$\frac{14}{51}$	$\frac{1017}{543}$	$\frac{687}{1017}$	$\frac{525}{694}$	$\frac{449}{607}$	$\frac{306}{545}$
	32	$\frac{3}{3}$	$\frac{2}{14}$	$\frac{2}{6}$	$\frac{2}{5}$	$\frac{3}{4}$	$\frac{42}{33}$	$\frac{20}{102}$	$\frac{15}{64}$	$\frac{14}{51}$	$\frac{14}{35}$	$\frac{875}{478}$	$\frac{388}{975}$	$\frac{442}{663}$	$\frac{316}{548}$	$\frac{165}{484}$
16	1	$*$	$*$	$*$	$*$	$\frac{3}{3}$	$*$	$*$	$*$	$*$	$\frac{25}{18}$	$*$	$*$	$*$	$*$	$\frac{559}{286}$
	4	$*$	$*$	$*$	$\frac{2}{13}$	$\frac{2}{13}$	$*$	$*$	$*$	$\frac{17}{104}$	$\frac{12}{100}$	$*$	$*$	$*$	$\frac{380}{841}$	$\frac{220}{838}$
	8	$*$	$*$	$\frac{3}{8}$	$\frac{3}{6}$	$\frac{6}{6}$	$*$	$*$	$\frac{21}{99}$	$\frac{15}{69}$	$\frac{10}{61}$	$*$	$*$	$\frac{490}{771}$	$\frac{277}{556}$	$\frac{226}{534}$
	16	$*$	$\frac{2}{13}$	$\frac{3}{6}$	$\frac{2}{4}$	$\frac{3}{4}$	$*$	$\frac{17}{104}$	$\frac{15}{69}$	$\frac{9}{41}$	$\frac{10}{32}$	$*$	$\frac{378}{849}$	$\frac{276}{551}$	$\frac{234}{394}$	$\frac{160}{332}$
	32	$\frac{3}{3}$	$\frac{3}{13}$	$\frac{3}{6}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{25}{18}$	$\frac{12}{100}$	$\frac{10}{61}$	$\frac{10}{32}$	$\frac{10}{20}$	$\frac{574}{287}$	$\frac{221}{843}$	$\frac{233}{530}$	$\frac{164}{344}$	$\frac{88}{288}$

TABLE 2: The execution time, rounded up to the nearest second, of the algorithm on an $M \times L$ decomposition with P processors. Results corresponding to minimal and maximal interfacial subdomains are given above and below the line, respectively. A $*$ indicates that the computational load of Step 2 cannot be balanced at the first level of parallelisation.

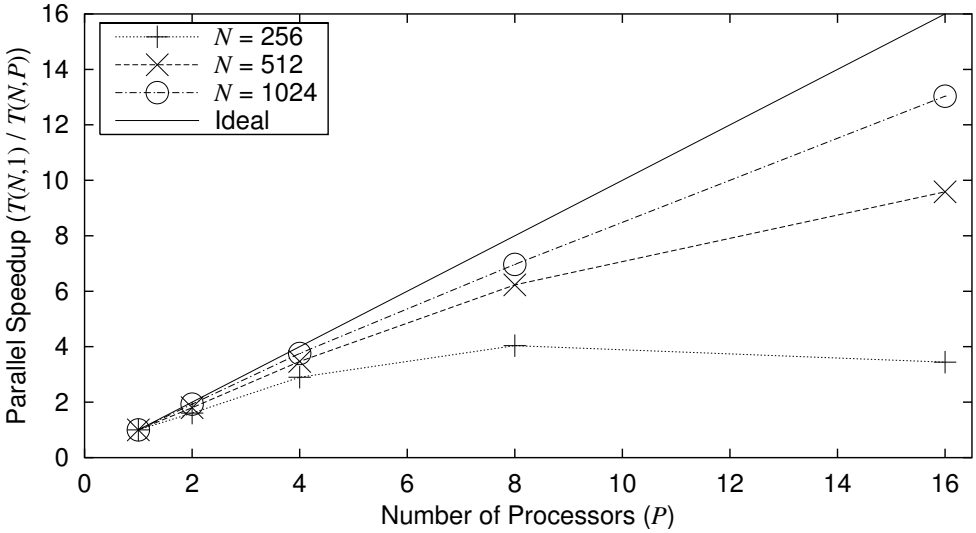

FIGURE 2: The parallel speedup for three mesh sizes N.

domain decomposition. With one processor, the (serial) code executes in 6.89 seconds. With two processors, the linear algebra of each step is shared equally but data must be transferred between the two processors before each step. In fact, each global iteration entails the inter-processor transfer of 4928 blocks of data. These blocks are buffered and sent as 6 MPI communications. This cost of composing and sending the messages more than offsets the reduced computational workload per processor. Thus, on two processors the algorithm executes in 7.60 seconds, and the best decomposition is that of 4×32, minimal interfacial subdomains.

As the global problem size increases to $N = 512$ and $N = 1024$, the linear algebra becomes increasingly significant and the communication overhead reduces correspondingly. Thus, for each decomposition we observe a reduction in execution time as the number of dedicated processors increases.

Given the serial performance of the algorithm, we quantify the parallel

speedup of our implementation. For a given mesh size N and number of processors P, let $T(N, P)$ be the minimum execution time over all domain decompositions. In Table 2, $T(N, P)$ is the smallest time in the major cell corresponding to N and P. We define the parallel speedup of the algorithm by the ratio $T(N, 1)/T(N, P)$. This is plotted in Figure 2 together with the ideal speedup. With $N = 256$, the parallel speedup is greatest for $P = 8$; with $P = 16$ the extra capacity for computation is negated by the necessary communication. However, as N increases to 512 and 1024 the linear algebra becomes increasingly significant and the communication overhead is justified.

5 Conclusion

We have considered the nonlinear reaction-diffusion problem and described a parallel implementation of the box-domain decomposition algorithm from [2]. Between any two processors, the necessary inter-subdomain transfers are buffered and sent as one MPI message. The parallel scale-up of the implementation improves with increasing N. On a 1024×1024 mesh, the optimal decomposition for any number of processors is the 32×32, minimal interfacial subdomain decomposition. On 16 processors we observed a parallel speedup of 13.04 which translates to a computational efficiency of 81.5%.

References

[1] I. Boglaev. On monotone iterative methods for a nonlinear singularly perturbed reaction-diffusion problem. *J. Comput. Appl. Math.*, 162:445–466, 2004. http://dx.doi.org/10.1016/j.cam.2004.02.010 C292

[2] I. Boglaev and M. P. Hardy. Monotone finite difference domain decomposition algorithms and applications to nonlinear singularly

perturbed reaction-diffusion problems. *Adv. Difference Eqns.*, (in press). http://www.hindawi.com/journals/ade/forthcoming/S1687183905409048.html C292, C293, C295, C298, C302

[3] J. J. H. Miller, E. O'Riordan, and G. I. Shishkin. *Fitted numerical methods for singular perturbation problems.* World Scientific, Singapore, 1996. C292

[4] C. Pao. Monotone iterative methods for finite difference system of reaction-diffusion equations. *Numer. Math.*, 46(4):571–586, 1985. http://dx.doi.org/10.1007/BF01389659 C292

[5] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual method for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986. http://locus.siam.org/SISC/volume-07/art_0907058.html C295