CHAPTER 30

Parallel Domain Decomposition and the Solution of Nonlinear Systems of Equations*

William D. Gropp†
David E. Keyes‡

Abstract. Many linear systems arise as subproblems in the solution of nonlinear equations, either as part of a simple fixed-point or a Newton's method iteration. This paper considers the use of domain decomposition techniques for the solution of these linear problems in the context of solving a multicomponent system of nonlinear equations on two types of parallel processors. One of the computations is drawn from fluid dynamics and includes locally refined grids. Such problems require great computational resources, and domain decomposition seems to offer a way to efficiently solve these problems on computers with significant parallelism. The domain decomposition approach used is as in Gropp and Keyes, modified to achieve better parallelism and to reduce the computational work.

1. Introduction. The work described in this contribution takes the domain decomposed Krylov method described in [9, 10] and extends it to systems of equations arising from the solution of nonlinear equations. Since the problems of interest are not self-adjoint, the conjugate gradient method does not apply, and there is little reason to use a symmetric preconditioner. The domain decomposition approach used here is a block triangularly preconditioned GMRES used by the authors in previous studies, requiring first a solve on a coarse grid of cross-points, then a set of independent interface solves, then a set of independent subdomain solves. While it is not optimal, it has in practice represented an efficient compromise of less work and data exchange per iteration for more iterations. Asymptotically in the reciprocal of the mesh parameter, the "optimal" methods, such as those described in [1] and [3], are to be preferred. For practical problem sizes, simpler approaches may be more efficient. In particular, the optimal methods described in [1, 3] involve two subdomain solves per subdomain for each application of the preconditioner; this requirement makes the "constant" multiplier in the time complexity of this method roughly twice as large as that of a method using a single subdomain solve. For the additive methods described in [5], the additional overlap required of the subdomains also increases the constant multiplier in the asymptotic work estimate. For many problems with modest accuracy requirements such as linear subiterations in a nonlinear problem, these inveigh against their optimal convergence rates. It is this problem domain that we consider in this paper.

---

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne Illinois 60439. The work of this author was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-ENG-38.

‡Department of Mechanical Engineering, Yale University, New Haven, Connecticut 06520. The work of this author was supported by the National Science Foundation under Contract number ECS-8957475, by the IBM Corporation, and by the 3M Company.

We also discuss the prospects for the use of extensive parallelism (more than 100 processors) in domain decomposition, and we provide some experimental and analytic estimates of the sources of inefficiency in parallel domain decomposition algorithms on this scale. In addition, we consider an approximate cross-point solver as a method for improving the parallel efficiency of these algorithms.

Previous Work. A number of authors have considered the application of domain decomposition methods to parallel computers (e.g., [6, 7, 8, 11, 13]). Others have considered the solution of nonlinear equations by these methods, focusing either on problems generated by a Newton-like solution algorithm (e.g., [12]) or generated by a time-splitting linearization (e.g., [4]).

A Brief Description of the Method. A typical division of a domain into subdomains (or *tiles*) is shown in Figure 1. The matrix corresponding to a five-point discretization of a PDE over this domain looks like

$$A = \begin{pmatrix} A_I & A_{IB} & 0 \\ A_{BI} & A_B & A_{BC} \\ 0 & A_{CB} & A_C \end{pmatrix},$$

where

$$\begin{aligned} A_I &= \text{block diagonal tile interiors} \\ A_{IB} &= \text{coupling to tile interiors from interfaces} \\ A_{BI} &= \text{coupling to interfaces from tile interiors} \\ A_B &= \text{block diagonal interfaces} \\ A_{BC} &= \text{coupling to interfaces from cross-points} \\ A_{CB} &= \text{coupling to cross-points from interfaces} \\ A_C &= \text{diagonal cross-points} \end{aligned}$$

As our preconditioner, we take a block triangular approximation to A:

$$B = \begin{pmatrix} \tilde{A}_I & \tilde{A}_{IB} & 0 \\ 0 & \tilde{A}_B & \tilde{A}_{BC} \\ 0 & 0 & \tilde{A}_C \end{pmatrix},$$

where, except for $\tilde{A}_C$, the "tildes" represent approximations (so $\tilde{A}_I$ is an approximation to A_I, such as a fast Poisson solver in a variable coefficient diffusion problem, or possibly an incomplete factorization). Note that $\tilde{A}_I$ is block diagonal, so its inverse is a highly parallelizable operation. Likewise, $\tilde{A}_B$ will typically be block diagonal, with block sizes equal to the number of interior degrees of freedom along each interface. However, $\tilde{A}_C$ is a coarse grid discretization of the original operator and thus *lacks* the simplicity of A_C. For it, we employ a direct banded elimination routine (a sparse elimination or an iterative procedure could be used as well). Note that only one set of subdomain

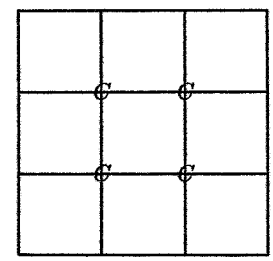

Figure 1: A sample decomposition of a domain. C denotes a cross-point; lines are tile-tile boundaries.

solves per iteration is needed to apply B^{-1}. As an aside pertaining to efficient implementation, we note that if $\tilde{A}_I = A_I$ and $\tilde{A}_{IB} = A_{IB}$, then the GMRES orthogonalization steps can be computed at a lower cost, since, as is readily verified, the rows of AB^{-1} corresponding to the interiors of the subdomains form an identity matrix.

In this paper, we solve systems of equations using the natural generalization of the scalar method described in [9]. This method relies on the form of interface solver. Of the methods considered in [9], the "tangential" preconditioner has an obvious generalization to systems of equations; we simply drop the normal derivatives, and solve the coupled system of equations along the interfaces.

2. Solving the Nonlinear Equations. The nonlinear equations are represented either as

$$F(u) = 0$$

r as the special case

$$A(u)u - b = 0.$$

n either case, the linear equations are solved by using GMRES preconditioned with the domain-ecomposed preconditioner described above, with the outer loop (the nonlinear iteration) being ither fixed point ($A(u^n)u^{n+1} = b$) or Newton ($u^{n+1} = u^n - F'(u^n)F(u^n)$).

In our experiments, the Jacobian for the Newton iteration is computed by analytically differentiating F (dropping high-order terms) and discretizing the result, rather than forming the Jacobian from differences of F.

Many optimizations and modifications may be applied; some are discussed in Brown and Saad [2]. While any production code should incorporate such refinements, for the purposes of this paper it is enough to implement the simplest version of these methods. Any improvements will likely not involve any new *kind* of work, only different (and more efficient, one hopes) application of the same kind of work. Thus, the *parallel* efficiencies of these methods will not be significantly affected by improvements to the algorithm.

It is important to note that we cannot solve only the linear part in parallel. Not only does Amdahl's law arise, but the cost of collecting and distributing the linear problems is high.

Sample nonlinear problems. We consider two classic nonlinear problems. The first is the Bratu problem

$$-\nabla^2 u + ce^u = 0 \tag{1}$$

on the unit square with homogeneous Dirichlet boundary conditions. The second is the streamfunction/vorticity formulation from computational fluid dynamics:

$$\begin{aligned} \nabla^2\phi + \omega &= 0 \\ -\frac{1}{Re}\nabla^2\omega + \left(\frac{\partial\phi}{\partial y}, -\frac{\partial\phi}{\partial x}\right)\cdot\nabla\omega &= 0. \end{aligned} \tag{2}$$

The problem domain is the step shown in Figure 2. The boundary conditions are ϕ constant along the walls and $\omega = -\frac{\partial^2\phi}{\partial n^2}$ along the walls, $\phi = y^2 - \frac{1}{3}y^3$ and $\omega = 2(y-1)$ along the in-flow boundary, and $\frac{\partial\omega}{\partial n} = 0$ and $\frac{\partial^2\phi}{\partial n^2} = 0$ on the out-flow edge.

Parallel Computing Issues. Our previous results [10] showed that the inefficiency in a parallel domain decomposition program comes from three main sources: the cross-point solve, the global dot products used in the GMRES algorithm, and the neighbor communication used in the matrix-vector multiplications and the preconditioner solves.

For modest to large numbers of processors, the first of these, the cross-point solve, is the dominant cause of loss of parallel efficiency [8].

Table 1 shows the results from different numbers of subdomains for a diffusion problem on an L-shaped region (Problem 8 in Table 4) on an Intel iPSC/2-SX. This machine has 4 megabytes of memory per node and roughly 0.5 megaflops per node of computing power. All computations were done in double precision. These results show that, while the best parallel speedups are achieved for the smallest numbers of tiles, the best elapsed time occurs when the size of the cross-point system and the size of an individual tile are comparable. Basically, a smaller number of tiles means that each individual tile is larger, and hence the interior solve on that tile is more expensive. This reduces the fraction of time that is spent coordinating the parallel computation, and thus gives a higher speedup. These effects may be modeled; more details are given found in [8]. These results confirm that we will need to manage a relatively large cross-point system if we are to get the best elapsed time for a computation.

Parallelizability of Nonlinear Solves. The outer loop of the nonlinear solve, whether it is a simple fixed-point iteration or a sophisticated Newton-like method, is as parallelizable as the inner (linear) solves. In particular, the outer nonlinear solve involves three steps (besides the inner solve):

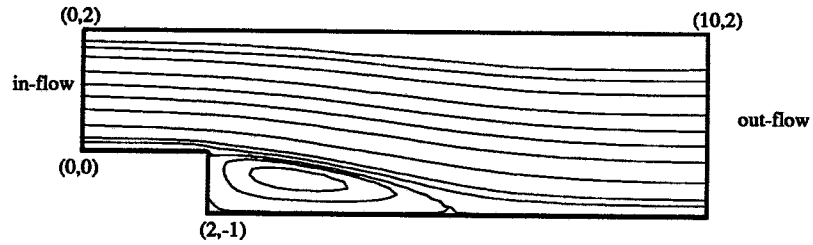

Figure 2: Domain for problem (2)

Table 1: Times (in seconds) and speedups for Problem 8 on different numbers of processors, showing the detrimental effects of the cross-point solves. Note also that the *smallest* cross-point system was not the best in terms of overall time, but was the best in terms of parallel speedup.

		Factor		Solve		Total	
Mesh	p	Time	Rel. Sp.	Time	Rel. Sp.	Time	Rel. Sp.
48 tiles	4	11.61		54.91		66.52	
each	8	5.94	1.95	28.61	1.92	34.55	1.93
16 × 16	16	3.13	1.90	15.30	1.87	18.43	1.87
192 tiles	8	2.28		16.95		19.23	
each	16	1.71	1.33	9.91	1.71	11.62	1.65
8 × 8	32	1.42	1.20	6.37	1.56	7.79	1.49
	64	1.28	1.11	4.67	1.36	5.95	1.31
768 tiles	16	15.79		19.92		35.71	
each	32	15.73	1.00	14.62	1.36	30.35	1.18
4 × 4	64	15.70	1.00	11.83	1.24	27.53	1.10

Table 2: Results from Encore 320 for the problems in (1) and (2). The "Bratu" results are for a Newton iteration and the "Step" results are for a fixed-point iteration. t is the number of tiles per unit length, and m is the number of subintervals along a tile side.

Problem	t	m	p	Total time	Speedup	Nonlinear time	Speedup
Bratu	8	5	1	51.4		7.2	
			2	26.5	1.9	3.7	1.9
			4	14.4	3.6	2.0	3.6
			8	8.3	6.2	1.2	6.0
Bratu	16	5	1	726		31.3	
			2	373	1.9	16.4	1.9
			4	197	3.7	8.6	3.6
			8	109	6.7	5.0	6.3
Bratu	5	16	1	142		106	
			2	74	1.9	55.8	1.9
			4	41	3.5	31.1	3.4
			8	21	6.8	17.5	6.1
Step	1	5	1	4378		225	
			2	2247	1.95	115	1.96
			4	1202	3.6	61	3.7
			8	699	6.3	38	5.9

- computation of residual vector (local communication),
- recomputation of coefficients for the problem *and the preconditioner* (all local, except for cross-point), and
- convergence test (global dot product).

No term here dominates the costs of GMRES solves. In particular, the preprocessing cost for the cross-point solve (when done by a direct method) involves the same sort of global communication as is involved in the solution phase. Note that the ratio of computation to communication is different in these two phases, but since the granularity of this subproblem is small, the overall efficiencies are similar.

Results of Parallel Computation of the Nonlinear Problems. As Table 2 shows, the parallel speedups of the part of the computation related to the nonlinear solves (computation of the residual, construction of the preconditioners, and the convergence tests) are nearly identical to those of the inner linear solves (note that in most cases the nonlinear solve is a small part of the computation, and the linear solves makes up most of the effort). A similar result was obtained with a distributed-memory version of our code on an Intel iPSC/2.

For the problems shown in Table 2, the coefficient c of the Bratu problem was 1, the Reynolds number for the "Step" problem was 100, and the numbers of unknowns were 1681, 6561, 6561, and 1532, respectively.

Parallelizing the Cross-Point Solver. Since the cross-point solver is a bottleneck in a parallel domain decomposition code, we discuss how it may be made more efficient. The obvious way to reduce the cost of the cross-point solver, by reducing the number of subdomains, is generally untenable. This limits the number of subdomains and hence, if subdomains are not further subdivided among processors, limits the amount of parallelism. Our target is a parallel computer with hundreds to thousands of processors, so this is not an option. Further, if the subdomains are of different (computational) sizes, resulting, for example, from differing amounts of refinement, then in order to balance the work load, some processors will need to process more than one subdomain. Thus, on the grounds of parallel load balancing alone, apart from the significant benefit in terms of the convergence rate, the cross-point system will be relatively large.

There are several possibilities to consider for faster and more parallel cross-point solvers. These represent different compromises between computation, communication, and quality of the preconditioner. Some choices are

1. sparse, direct (less computation, more communication);
2. iterative (less computation and communication for a loose convergence tolerance); and
3. concurrent solution of the cross-point system.

We will consider the second of these. The third is inherent in the function space decomposition of the additive Schwarz method [5], and is also advocated in [3].

Complexity Estimates. As two limiting cases, we consider a banded solve for the cross-point system (an expensive procedure in both computation and communication time) and a wavefront-ordered Gauss-Seidel-like iterative method (very cheap in both terms). Of course, the banded solve gives the exact solution to the cross-point system, and the iterative method only an approximate value. The questions are: (1) Is the iterative solver, at the relatively fine grain of the cross-point system, significantly cheaper per iteration than the banded solve? and (2) What penalty do we pay in terms of iteration count for using the iterative solver? The first question may be answered by looking at the time complexities of the parallel versions of these two approaches. We also consider two other limiting cases: computing on one processor (trading communication for computation) and computing on all processors.

Our notation is

f = time for a floating point operation

s = time for an I/O startup or synchronization

I_b = GMRES iterations for the banded solve preconditioner

I_{gs} = GMRES iterations for the Gauss-Seidel (GS) solve preconditioner

p = number of processors.

We consider an $n \times n$ mesh of tiles. Here, "startups" refers to message startup times on a distributed-memory computer and synchronizations to barriers on a shared-memory computer. For simplicity, we are neglecting the size of the data transfer in the cost of communication.

If all processors are involved in the computation, then the banded solve takes about

$$n^4 f/p + 2n^2 s + I_b(2n^3 f/p + 2n^2 s)$$

time, and the wavefront GS takes

$$I_{gs}(M5n^2 f/p + 4(M+n)s)$$

for M inner iterations of the GS method. Note that in the case of the banded solver, much of the time is spent in computing the factors (the term in n^4).

A feature common to both of these approaches is that there is a fair amount of communication involved. If the cost of communication is high enough, it can actually be more efficient to collect the data on a single processor (or a smaller number of processors) and compute it there. In fact, some communication can be saved by having each processor do the *same* computation after all have exchanged data. In this case, the banded solve takes roughly

$$n^4 f + I_b(2n^3 f + s \log p),$$

and the wavefront GS takes

$$I_{gs}(M5n^2 f + s \log p).$$

Note that in this case, $M < n$ for the GS solver to be better than the banded solver (assuming that $I_b \approx I_{gs} > n$, so that the setup or factorization cost is subdominant).

Using these formulas, we can answer some questions about the relative effectiveness of these approaches. First, we will compare the distributed to the local approaches.

For distributed version of GS to be faster than local GS, we must have.

$$\frac{s}{f} < \frac{M}{4(M+n) - \log p} 5n^2 \left(1 - \frac{1}{p}\right),$$

Table 3: Timings of wavefront GS on an iPSC/2. The mesh is $n \times n$.

p	n	Time	Speedup
1	32	3.8	
4		2.3	1.7
16		1.6	2.4
64		0.92	4.1
1	64	30.2	
4		14.2	2.1
16		8.8	3.4
64		5.4	5.6

Table 4: The test problem suite (from [9]). Values of f are chosen to match the given solution. The values of c for Problems 8–10 are $c = 0$, -10, and 10, respectively. Boundary conditions are Dirichlet unless otherwise noted.

Test	Equation	Boundary	Solution
4	$-\nabla^2 u + c\frac{\partial u}{\partial y} = f$	$u_x(x,1) = 0$	$u = \sin(\pi x)\sin(\frac{\pi y}{2})$
5	$\frac{\partial}{\partial x}\left(e^{xy}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(e^{-xy}\frac{\partial u}{\partial y}\right) - \frac{u}{1+x+y} = f$		$u = e^{xy}\sin(\pi x)\sin(\pi y)$
6	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial}{\partial y}\left((1+y^2)\frac{\partial u}{\partial y}\right) - \frac{\partial u}{\partial x} - (1+y^2)\frac{\partial u}{\partial y} = f$	$u - \frac{\partial u}{\partial n} = g$	$u = 0.135(e^{x+y} + (x^2-x)^2\log(1+y^2))$
8–10	$-\nabla^2 u + \frac{c}{r}\frac{\partial u}{\partial r} = 0$		$u = r^\alpha \sin(\frac{2}{3}(\theta - \frac{1}{2}\pi))$ $r=\sqrt{(x-1)^2+(y-1)^2}$ $\theta=\arg((x-1)+i(y-1))$, $0\le\theta<2\pi$

where we have assumed a hypercube or other communication network with $\log p$ time to distribute data from all processors to all processors. For banded version (linear pipe with log scatter) to be faster than local banded, we must have

$$\frac{s}{f} < \frac{1}{2n^2 - \log p} 2n^3 \left(1 - \frac{1}{p}\right) \approx n\left(1 - \frac{1}{p}\right).$$

Finally, if we take the fastest of these two for the size of p and n that characterize our current problem sets, we see that for distributed GS to be faster than local banded,

$$\frac{s}{f} < \frac{1}{4(M+n) - \log p} n^2 \left(2n - \frac{5M}{p}\right).$$

Here we have assumed that $I_{gs} = I_b$, since we expect $I_{gs} \geq I_b$, and the assumption of equality gives us a bound (if s/f exceeds the right-hand side instead, then the local banded will be better).

Table 3 shows the observed speedups for the wavefront ordered algorithm running on the indicated number of processors. The subdomains were assigned to processors in such a way as to reduce amount of communication. These timings are samples; they could be tuned to produce slightly higher speedups. The message here is that a factor of 2 or even 4 improvement in the times will still yield a relatively slow solver (at 64 processors, an efficiency of only 24–30%).

Experimental Iteration Counts. In interpreting the complexity analysis, we assumed that $I_{gs} = I_b$. In fact, we expect the iteration counts for the approximate cross-point solver to be larger than those of the direct solver. Since the question of "how much larger" is difficult to attack analytically, we performed a series of numerical experiments, using the problem suite introduced in [9]. The problems that we used in our experiments are shown in Table 4.

Table 5: Iteration count results for problem suite in Table 4. Each subdomain is 4×4 in size. The number of subdomains (tiles) along a side of the domain is given by the column labeled t. GMRES was not restarted in these examples. These times are for runs on a single SPARCStation 1.

Problem	t	Direct		GS	
		Iterations	Time	Iterations	Time
4	16	19	34	31	66
4	32	19	160	45	509
5	16	22	42	36	83
5	32	23	203	—	—
6	16	17	30	52	151
6	32	15	124	—	—
8	16	12	14	18	23
8	32	11	65	28	180
9	16	16	21	19	25
9	32	15	88	24	142
10	16	13	16	13	15
10	32	10	58	14	67

Table 5 shows that the simple GS iteration is rarely advantageous. In even the simplest problem (Problem 4), the number of GMRES iterations increased substantially, dramatically increasing the solution times. For Problems 5 and 6, the GMRES iterations were making so little progress that they had not converged after 500 iterations. In contrast, for Problem 10, the GS iteration was actually comparable to the direct solution in terms of iteration count and overall time. These results suggest that a suitable approximate cross-point system solver may be appropriate for problems with special structure. Combing these results with the complexity estimates above, we can say that for the general case, any approximate cross-point solver must be more accurate than the GS method but must be nearly as inexpensive. A fast multigrid solver might be employed.

3. Conclusions. Our results show that cross-point solution remains a problem for massively parallel domain decomposition algorithms. Computationally, the relatively small size of the cross-point problem makes it difficult to develop an efficient parallel algorithm, since the amount of floating point work relative to the amount of communication is small. We have showed experimentally that a simple approximate cross-point solver is rarely suitable; we leave it as a challange to determine how accurate a solution of the cross-point system is must ensure the optimality properties of the various flavors of domain decomposition methods. The fully additive methods may provide a way around this difficulty, as the cross-point problem may be computed in parallel with the interior solves.

We have also shown that domain decomposition is an effective preconditioner for the parallel solution of problems arising from (some) nonlinear equations, in the case where a relatively small number (less than 100) of processors is used. In particular, the cost of the steps associated with the nonlinear solve, whether it be a simple fixed point or Newton method, parallelizes at least as well as steps in the domain decomposition algorithm for the inner linear problem. The sequence of linear problems that we encountered in our model problems were all efficiently solved by our domain decomposition approach.

References

[1] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The Construction of Preconditioners for Elliptic Problems by Substructuring, IV,* Math. Comp., 53 (1989), pp. 1–24.

[2] P. Brown and Y. Saad, *Hybrid Krylov Methods for Nonlinear Systems of Equations,* SIAM J. Sci. Statist. Comput., 11/3 (1990), pp. 450–481.

[3] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *Convergence Rate Estimate for a Domain Decomposition Method,* 1990. (Manuscript).

[4] E. J. Dean, R. Glowinski, and O. Pironneau, *Iterative Solution of the Stream Function-Vorticity Formulation of the Stokes Problem. Applications to the Numerical Simulation of Incompresible Viscous Flow,* Technical Report UH/MD-89, Department of Mathematics, University of Houston, August 1990.

[5] M. Dryja and O. B. Widlund, *An Additive Variant of the Schwarz Alternating Method for the Case of Many Subregions,* Technical Report TR 339, NYU, Courant Institute, December 1987.

[6] D. Goovaerts, *Domain Decomposition Methods for Elliptic Partial Differential Equations,* Ph.D. Thesis, Catholic University in Leuven, May 1990.

[7] A. Greenbaum, A. C. Li, and H. Z. Chao, *Parallelizing Preconditioned Conjugate Gradient Algorithms,* Comp. Phys. Comm., 53 (1989), pp. 295–309.

[8] W. D. Gropp and D. E. Keyes, *Domain Decomposition on Parallel Computers,* Impact Comput. Sci. Eng., 1 (1989), pp. 421–439.

[9] ———, *Domain Decomposition with Local Mesh Refinement,* Technical Report RR-726, Yale University, Dept. of Comp. Sci., August 1989.

[10] ———, *Parallel Performance of Domain-decomposed Preconditioned Krylov Methods for PDEs with Adaptive Refinement,* Technical Report MCS-P147-0490, Mathematics and Computer Science Division, Argonne National Laboratory, May 1990.

[11] D. E. Keyes and W. D. Gropp, *A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and their Parallel Implementation,* SIAM J. Sci. Stat. Comp., 8 (1987), pp. s166-s202.

[12] ———, *Domain Decomposition Techniques for the Parallel Solution of Nonsymmetric Systems of Elliptic BVPs,* Appl. Num. Math., 6 (1990), pp. 281–301.

[13] D. Resasco, *Domain Decomposition Algorithms for Elliptic Partial Differential Equations,* Technical Report YALEU/DCS/RR-776, Computer Science Department, Yale University, May 1990.