SIAM J. SCI. COMPUT. © 2002 Society for Industrial and Applied Mathematics
Vol. 24, No. 1, pp. 335–358

STABILIZED EXPLICIT-IMPLICIT DOMAIN DECOMPOSITION METHODS FOR THE NUMERICAL SOLUTION OF PARABOLIC EQUATIONS*

YU ZHUANG† AND XIAN-HE SUN‡

Abstract. We report a class of stabilized explicit-implicit domain decomposition (SEIDD) methods for the numerical solution of parabolic equations. Explicit-implicit domain decomposition (EIDD) methods are globally noniterative, nonoverlapping domain decomposition methods, which, when compared with Schwarz-algorithm-based parabolic solvers, are computationally and communicationally efficient for each simulation time step but suffer from small time step size restrictions. By adding a stabilization step to EIDD, the SEIDD methods retain the time-stepwise efficiency in computation and communication of the EIDD methods but exhibit much better numerical stability. Three SEIDD algorithms are presented in this paper, which are experimentally tested to show excellent stability, computation and communication efficiencies, and high parallel speedup and scalability.

Key words. parallel computing, nonoverlapping domain decomposition, parabolic equation, globally noniterative method

AMS subject classifications. 65M55, 65Y05, 65Y20

PII. S1064827501384755

1. Introduction.

We report a class of stabilized explicit-implicit domain decomposition (SEIDD) algorithms [39] for the numerical solution of the initial boundary value problem of the parabolic equation

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} = Au(t,x)+f(t,x), & x\in\Omega, \quad t\geq 0, \\ u(t,x) = u_b(t,x), & x\in\partial\Omega, \quad t\geq 0, \\ u(0,x) = u^0(x), & x\in\Omega, \end{cases} \tag{1.1}$$

where A is the spatial differential operator

$$Au = \sum_{i=1}^{k} \frac{\partial}{\partial x_i}\left(a_i(x)\frac{\partial u}{\partial x_i} + b_i(x)u\right) + c(x)u, \tag{1.2}$$

Ω is a compact subset in $\mathbf{R}^k$ with $k=1,\ 2$, or 3, and the spatial variable $x = (x_i)_{i=1}^k$. The functions a_i and b_i are continuously differentiable, and c and f are continuous.

The explicit-implicit domain decomposition (EIDD) algorithms [3, 13, 14, 24, 27, 28] are globally noniterative, nonoverlapping methods, which are computationally and communicationally efficient for *each time step* when compared with Schwarz-type domain decomposition elliptic solvers incorporated into implicit temporal discretizations. However, EIDD methods suffer from either stability- or consistency-related time step size restrictions (see section 2), while Schwarz methods could maintain the good stability condition of implicit temporal discretization schemes. If the time-stepwisely efficient EIDD algorithms are freed from time step size restrictions, they will possess great potential for large-scale parallel simulations on distributed memory machines.

*Received by the editors February 8, 2001; accepted for publication (in revised form) January 18, 2002; published electronically July 16, 2002. This research was supported in part by NSF grant CCR-9972251 and by ONR under the PET program.

http://www.siam.org/journals/sisc/24-1/38475.html

†Computer Science Department, Texas Tech University, Lubbock, TX 79409 (zhuang@cs.ttu.edu).

‡Computer Science Department, Illinois Institute of Technology, Chicago, IL 60616 (sun@cs.iit.edu).

In this paper, we report a class of stabilized EIDD (SEIDD) algorithms [39]. In addition to a generic parallel SEIDD algorithm with the choices of the predictor, subdomain scheme and the stabilizer open to the algorithm implementors and users, we also present three specific algorithms of this SEIDD class as examples. One (denoted SEIDD1) is the stabilization of an EIDD algorithm due to Kuznetsov [24] which uses the forward Euler scheme as the interface boundary condition predictor and the backward Euler scheme for temporal discretization on the subdomains. Another (SEIDD2) is the stabilization of Dawson, Du, and Dupont's algorithm [14] by a stabilizer designed for the predictor. The third (SEIDD3) differs from SEIDD1 in that the subdomain temporal scheme is an approximate directional factorization of the backward Euler, which retains the same temporal accuracy but reduces the computation complexity of the subdomain solver to $O(N)$ for each time step on a spatial domain of N grid points. This linear complexity of the SEIDD3 algorithm also holds for nonselfadjoint problems whose discretized spatial operator A_h is a nonsymmetric matrix. By adding a stabilization step to the EIDD methods, the SEIDD algorithms exhibit much better stability. But more importantly for parallel computing, the stabilization is proven to add *zero* additional communication cost and very low computation cost to EIDD algorithms, yielding excellent parallel speedup and scalability confirmed by testings on SGI Origin 2000 computers for a wide range of parabolic problems from heat equation to convection-diffusion to nondissipative convection-diffusion.

This paper focuses on the description of the algorithms, the parallel speedup and efficiency analysis, and numerical experiments. The rigorous mathematical proof of the stability is still under investigation, but heuristic explanations are provided. The paper is organized as follows. A brief survey of domain decomposition algorithms for parabolic problems is provided in section 2. A generic parallel SEIDD algorithm is presented in section 3 together with analysis of computation and communication overhead of the stabilization, as well as the parallel speedup and efficiency. Three specific algorithms of SEIDD type are presented in sections 4 to 6. Section 7 contains the numerical experimental results, and section 8 gives the conclusion.

2. Existing domain decomposition algorithms.

2.1. The Schwarz algorithms. There exists a substantial literature in domain decomposition. The Schwarz alternating algorithms [6, 7, 8, 12, 18, 21, 22, 23, 33, 34] are globally iterative domain decomposition procedures for solving elliptic problems with domain-decomposition-based matrix splittings to enhance parallelism and localized treatment of irregular geometries. Here the term "globally" refers to the part of a solution process that is carried over the entire problem domain as opposed to solution processes for subdomain problems which could be either iterative or direct. For parabolic equations, after implicit temporal discretization, an elliptic equation of the form

$$(I - \Delta t A)u = r \tag{2.1}$$

needs to be solved for each time step. So the Schwarz algorithm can be used for the parallel solution of parabolic problems by solving the elliptic equation (2.1) in parallel. Cai [4, 5] used the Schwarz algorithm in this manner to solve parabolic problems. Since the Schwarz algorithm is a solver for the elliptic equation (2.1) resulting from temporal discretization, it has an advantage in preserving the unconditional stability of implicit temporal discretizations as long as the Schwarz solver is iterated until the solution error becomes small enough to have no influence on the stability of the temporal scheme. Such an accuracy requirement for preserving stability of the implicit temporal discretizations can still be done efficiently with the elimination of the requirement of a coarse level global solver when the time step size Δt is subject to a very modest condition $\Delta t \leq O(H^2)$, where H is the diameter of a subdomain.

2.2. Globally noniterative overlapping algorithms. In the parallel implementation of domain decomposition algorithms, when different subdomains are assigned to different processors, globally iterative methods incur repeated data transmission among processors. Since communication is much more time consuming per byte than computation (floating point operations), it is appealing to keep the global iterations to a small number.

In 1988, Kuznetsov [24] proposed a one-iteration overlapping Schwarz algorithm for the numerical solution of the elliptic equation (2.1) obtained from an implicit temporal discretization of parabolic problems (see [9, 25] for similar results). His algorithm utilizes the property that after A is spatially discretized into A_h, the entries in the matrix $(I - \Delta t A_h)^{-1}$ decay rapidly when their positions recede from the diagonal. Thus, entries far away from the diagonal are in fact dropped from $(I - \Delta t A_h)^{-1}$. Kuznetsov's elliptic solver has a good stability condition, but requires an overlap size of $O(\sqrt{\Delta t} \log \epsilon)$ for a local error tolerance of $O(\epsilon)$. Thus, his algorithm requires an overlap size of $O(\sqrt{h} \log(h^3))$ to reach a local accuracy of $O(\Delta t\, h^2)$ with $\Delta t = h$.

In 1998, Mathew et al. [31] gave an overlapping algorithm with a lesser requirement of overlap size than the Kuznetsov algorithm. The algorithm of Mathew et al. uses the decomposed domain to construct a partition of unity which consists of nonnegative *smooth* functions with each subdomain being associated with a member function. Each member function of the partition of unity is supported on and vanishes outside of the subdomain it is associated with. Then a splitting of the spatial operator A into

$$A = A_1 + A_2 + \cdots + A_p \tag{2.2}$$

is constructed using the member functions that make up the partition of unity. After splitting of the operator, they temporally discretize the equation using Douglas's multidimensional ADI method [15]

$$u^{n+1} = (I - \tfrac{\Delta t}{2} A_1)^{-1} \cdots (I - \tfrac{\Delta t}{2} A_p)^{-1} (I + \tfrac{\Delta t}{2} A_p) \cdots (I + \tfrac{\Delta t}{2} A_1) u^n, \tag{2.3}$$

with the directional components of A in Douglas's original alternating direction implicit (ADI) method replaced by the components in the operator splitting (2.2). The algorithm of Mathew et al. is not subject to a large overlap size requirement because they construct their operator splitting through smooth, compactly supported member functions of the partition of unity, which has another advantage, that is, predicting interface boundary conditions for the inverting operations in the ADI (2.3) is no longer required. This is due to the property that each member function of the partition of unity is smooth and vanishes at the boundary of the subdomain with which it is associated. Since ADI has a second order temporal accuracy, this domain decomposition algorithm has an $O\big((\Delta t)^3\big)$ local temporal accuracy. However, just like the original multidimensional ADI method, it loses its good stability when the operator is split into more than two noncommutative components. Thus the stability suffers when the number of subdomains goes above two.

2.3. Nonoverlapping domain decomposition algorithms. Since nonoverlapping algorithms have low computation and communication cost for each time step, they are appealing for large problems on massively parallel machines. In the following, we list some globally noniterative, nonoverlapping domain decomposition algorithms that are not of the EIDD type. The EIDD algorithms will be briefly described in the section that follows.

In 1991, Dryja [17] proposed a one-iteration solver for parabolic problems temporally discretized by the Crank–Nicolson scheme [10]

$$(I - \tfrac{\Delta t}{2}A)u^{n+1} = (I + \tfrac{\Delta t}{2}A)u^n. \tag{2.4}$$

In his algorithm, Dryja uses a red-black ordering of the subdomains and divides the subdomains into two groups, Ω_1 and Ω_2. Equation (2.4) is then solved alternately on the two groups of the subdomains with the Galerkin method

$$\begin{cases} \left\langle \frac{u^{n+1/2}-u^n}{\Delta t}, v \right\rangle = a_1\left(\frac{u^{n+1/2}+u^n}{2}, v\right) & \text{on } \Omega_1, \\ u^{n+1/2} \;=\; u^n & \text{on } \Omega_2, \end{cases} \tag{2.5}$$

$$\begin{cases} \left\langle \frac{u^{n+1}-u^{n+1/2}}{\Delta t}, v \right\rangle = a_2\left(\frac{u^{n+1}+u^{n+1/2}}{2}, v\right) & \text{on } \Omega_2, \\ u^{n+1} \;=\; u^{n+1/2} & \text{on } \Omega_1, \end{cases} \tag{2.6}$$

where $\langle u, v\rangle = \int_\Omega u(x)v(x)\,dx$ and $a_i\,(u, v) = \int_{\Omega_i} Au(x)\,v(x)\,dx$. Thus from a temporal discretization point of view, Dryja's solver is a fractional step method. When viewed as an elliptic solver, it is a one-iteration minimal overlapping (the closures $\overline{\Omega}_1$ and $\overline{\Omega}_2$ have an overlap of width h) multiplicative Schwarz algorithm. From (2.5) and (2.6) it can be easily derived that the algorithm is representable in the non-Galerkin form as

$$u^{n+1} = (I - \tfrac{\Delta t}{2}A_2)^{-1}(I + \tfrac{\Delta t}{2}A_2)(I - \tfrac{\Delta t}{2}A_1)^{-1}(I + \tfrac{\Delta t}{2}A_1)u^n,$$

where A_1 and A_2 are the restrictions of the spatially discretized operator A_h on Ω_1 and Ω_2, respectively. With the above representation, it is immediately seen that this algorithm is unconditionally stable, since for each $i = 1, 2$,

$$\|(I - \tfrac{\Delta t}{2}A_i)^{-1}(I + \tfrac{\Delta t}{2}A_i)\| \le e^{\omega\Delta t}$$

for some norm $\|\cdot\|$. However, as Dryja himself pointed out, this algorithm has a large global error of order $O(\sqrt{\Delta t} + h)$ when Δt is proportional to h.

An algorithm with better global error was proposed by Laevsky [26] for the entire domain decomposed into two subdomains, Ω_1 and Ω_2. His algorithm uses the Galerkin method for the elliptic ADI-solver [32, 41] of the Crank–Nicolson scheme

$$\begin{cases} \frac{u^{n+1/2}-u^n}{\Delta t/2} = A_1u^{n+1/2} + A_2u^n, \\ \frac{u^{n+1}-u^{n+1/2}}{\Delta t/2} = A_1u^{n+1/2} + A_2u^{n+1}, \end{cases} \tag{2.7}$$

where A_1 and A_2 are the restrictions of the matrix A_h on Ω_1 and Ω_2, respectively. The global error of this algorithm is of order $O(h + (\Delta t)^2 + \rho(1+\rho)^{1/2})$ with $\rho = (\Delta t)^2/h$. However, the term $\rho(1+\rho)^{1/2}$ has placed a restriction on the ratio of Δt and h.

The ADI method is unconditionally von Neumann stable and unconditionally convergent [16, 37, 38] for quasi-dissipative problems when the operator A is split into two components. However, when the operator is split into more than two noncommutative components, the ADI scheme loses unconditional von Neumann stability (the

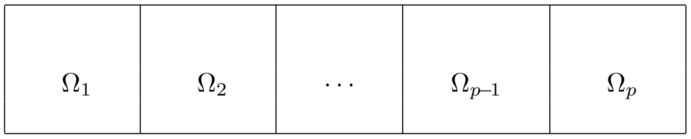

FIG. 1.

instability also mentioned in [28]) and requires the time step size to be of $\Delta t = O(h^2)$ for the multicomponent ADI scheme to converge. Thus when the entire domain is decomposed into more than two subregions, Laevsky's algorithm (2.7) not only loses its good stability condition, but the term $\rho(1+\rho)^{1/2}$ will enlarge its global error to order $O(1)$ as well.

Another algorithm of Laevsky and Rudenko [29] that is unconditionally convergent is a modification of the aforementioned Dryja's algorithm with a boundary treatment. This algorithm has a global error of $O(\Delta t + h)$, which is still low in spatial accuracy.

2.4. EIDD algorithms. One reason for the low accuracy of the nonoverlapping algorithms mentioned in the previous section is the lack of an adequately accurate boundary condition at the subdomain boundary points which are inside the entire problem domain. For instance, in (2.5) of Dryja's algorithm, the intermediate solution $u^{n+1/2}$ does not have an accurate interior boundary condition. An obvious solution is to use u^n to provide the interior boundary condition for $u^{n+1/2}$. This obviously introduces errors into the numerical solution.

Explicit-implicit algorithms have solved the problem of the availability of interior boundary conditions. In 1988, Kuznetsov [24] proposed an explicit-implicit scheme using a nonoverlapping domain decomposition as in Figure 1. The value of u^{n+1} on the interior boundary of the subdomains is first predicted using an explicit method. Then a stable implicit temporal discretization scheme can be applied to the equation on the subdomains, and the resulting elliptic equation on each subdomain can be solved independently using the predicted interface boundary conditions together with the exterior boundary conditions. When the forward Euler scheme is used as the interface boundary condition predictor and the backward Euler scheme is used for the temporal discretization of (1.1) on the subdomains, the explicit-implicit method can be represented as

$$u^{n+1} = (I - \Delta t A_2)^{-1}(I + \Delta t A_1)u^n, \tag{2.8}$$

where A_1 denotes the restriction of A_h on the interface boundary, and A_2 the restriction of A_h on the complement of interface boundary in the entire domain. However, the explicit predictor of the interface boundary condition causes numerical instability unless the time step size is restricted to $\Delta t = O(h^2)$.

An alternative explicit-implicit algorithm proposed by Dawson, Du, and Dupont [14] factorizes the interface boundary condition predictor $(I + \Delta t A_1)$ in (2.8) into

$$(I + \Delta t A_1) = (I - \Delta t A_1^y)^{-1}(I + \Delta t A_1^x)$$

for a domain decomposed as in Figure 1, where A_1^x denote the x-directional differentiation component of the operator A_1, and A_1^y the y-directional differentiation component. After the prediction of interface boundary conditions, the implicit backward Euler temporal scheme $(I - \Delta t A)u^{n+1} = u^n$ is applied to the system (1.1) on the subdomains, which can be solved independently on each subdomain. Thus, Dawson, Du, and Dupont's algorithm can be expressed as

$$u^{n+1} = (I - \Delta t A_2)^{-1}(I - \Delta t A_1^y)^{-1}(I + \Delta t A_1^x)u^n. \tag{2.9}$$

Using a wider spatial mesh $h_x > h$ in the x-direction for the predictor, the time step size is restricted to $\Delta t = O(h_x^2)$ instead of $\Delta t = O(h^2)$. Subject to the restriction $\Delta t \leq h_x^2/2$ for the two-dimensional heat equation, the global error of their algorithm is of $O(h^2 + \Delta t + 4\Delta t h_x + 2h_x^3)$. Thus taking $h_x = \frac{h^{2/3}}{2}$, the time step restriction for the algorithm (2.9) is $\Delta t = \frac{h^{4/3}}{8}$ with a good global error of $O(h^2) + O(\Delta t)$.

A penalized explicit-implicit algorithm proposed by Black [3] has remedied the stability related time step size restriction of the EIDD algorithms and achieved numerically verified unconditional stability. In the algorithm, Black [3] employed a Du Fort–Frankel-type scheme [19] as the explicit predictor. The Du Fort–Frankel scheme contains a penalty term, which, in a fashion similar to what Funaro did in [20], penalizes the nonsmoothness of the solution across the interface boundary. This penalized Du Fort–Frankel scheme achieves good numerical unconditional stability; however, it introduces an error term of $O(\frac{\Delta t}{h})^2$ for the heat equation, making the algorithm inconsistent unless $\Delta t/h \to 0$. Thus consistency comes only after paying a price of restricting time step size to an order of $O(h^2)$ to achieve a first order temporal accuracy, a restriction quantitatively similar to, though qualitatively different from, the restriction on the algorithm of Kuznetsov. Though not explicitly given in the paper, Black's algorithm is supposed to have a global error of $O(h^2) + O(\Delta t) + O\left((\frac{\Delta t}{h})^2\right)$.

There are other papers by Laevsky and his colleagues [27, 28] on explicit-implicit algorithms, but unconditional stability is not attained and the global errors are $O(h) + O(\Delta t)$ or larger.

2.5. Corrected EIDD methods. Besides Dawson, Du, and Dupont's factorization and Black's penalty method, there are other efforts to improve the stability of EIDD algorithms. J. Zhu [36] investigated an implicit correction technique for Kuznetsov's EIDD method for the one-dimensional heat equation and compared its stability results experimentally with several other methods. One author of this paper considered correction techniques in his Ph.D. thesis proposal and implemented an implicit correction method for Kuznetsov's EIDD method [40]. Daoud, Khaliq, and Wade used implicit correction [11] for Dawson, Du, and Dupont's method [14] applied to general parabolic problems with proven stability when Δt is restricted to $O(h_x^2)$, where h_x denotes the spatial mesh size used in the predictor in the direction across the interface boundary.

The idea of implicit correction is to replace the explicitly computed interface boundary condition by a new solution on the interface boundaries computed by an implicit method. There is a difference between the implicit correction and the stabilization which is advocated in this paper. From the operational point of view, stabilization can be considered a special case of correction, but with the aim to "stabilize" the errors on both the interface boundaries and the subdomains through a special design of the stabilizer to pair with the explicit predictor. The difference between stabilization and correction is illustrated in section 5.

3. A generic parallel SEIDD algorithm.

To ease the time step size restriction of the EIDD algorithms, a class of SEIDD algorithms [39] is introduced. The generic parallel version of the SEIDD methods is described below.

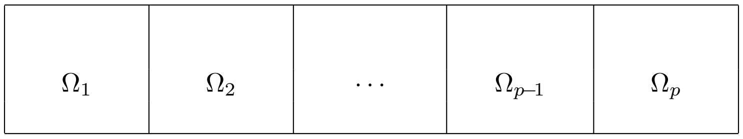

FIG. 2.

The entire domain Ω is divided into p subdomains $\Omega_1, \Omega_2, \ldots, \Omega_p$ (e.g., as in Figure 2) with interface boundaries denoted by Γ. The complement of the interface boundary is the subdomains whose union is denoted by Γ^c, namely, $\Gamma^c = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_p$. Thus, $\Omega = \Gamma \cup \Gamma^c$. For the numerical solution of (1.1), we choose a discrete spatial grid Ω_h and discretize the original problem spatially into

$$\begin{cases} \frac{\partial u(t,x)}{\partial t} = A_h u(t,x) + f(t,x), & x \in \Omega_h, \quad t \geq 0, \\ u(t,x) = u_b(t,x), & x \in \partial\Omega_h, \quad t \geq 0, \\ u(0,x) = u^0(x), & x \in \Omega_h, \end{cases} \tag{3.1}$$

where $\partial\Omega_h$, defined as $\partial\Omega \cap \Omega_h$, is the set of exterior boundary points on the discrete domain. Then we define the partitioning of the discrete domain Ω_h simply by inheriting the partitioning of the original domain:

$$\begin{cases} \Omega_{h,i} = \Omega_h \cap \Omega_i \quad \text{for } i = 1,\ 2, \ldots,\ p, \\ \Gamma_h = \Omega_h \cap \Gamma, \\ \Gamma_h^c = \Omega_h \cap \Gamma^c = \Omega_{h,1} \cup \Omega_{h,2} \cup \cdots \cup \Omega_{h,p}. \end{cases}$$

We denote the discrete interface boundary between subdomains Ω_i and Ω_j by $\Gamma_h^{i,j}$ for $i < j$ ($\Gamma_h^{i,j}$ could be an empty set) and denote the ith processor by p_i. Now a generic parallel SEIDD algorithm for computing the solution u_h^{n+1} at the $(n+1)$st time step from the current nth time step is given below.

THE PARALLEL SEIDD ALGORITHM.

0. Assign subdomain Ω_i and interface boundary $\Gamma_h^{i,j}$ to p_i.
1. Compute u_h^{n+1} at Γ_h using an explicit scheme. Then pass from p_i to p_j the newly predicted u_h^{n+1} on $\Gamma_h^{i,j}$.
   These computed data provide the interface boundary conditions.
2. Compute u_h^{n+1} on the subdomains Γ_h^c using any unconditionally stable scheme with the interface boundary conditions computed at step 1 as boundary conditions. Then pass part of the just computed data of u^{n+1} on the subdomain from p_j to p_i for the stabilization operation at the next step.
   Using any unconditionally stable temporal scheme results in an elliptic equation to be solved. The solution of the elliptic equation can be carried out mutually independently on the subdomains and thus in parallel.
3. Throw away the interface boundary condition computed at step 1 and bring back u^n on Γ_h. Then implicitly recompute u_h^{n+1} on Γ_h, using solution data u_h^{n+1} on Γ_h^c nearby as boundary conditions.
   Go back to step 1 for the next time step iteration.

In step 1 of the algorithm, we do not pass any part of u^n from p_j to p_i before the explicit computation of the interface boundary condition (IBC) u^{n+1}. The computation of the IBC does need those data. However, since processor p_i already received them from p_j at step 2 of the previous time step and no update has been performed on the data on the subdomains, no data transfer is necessary.

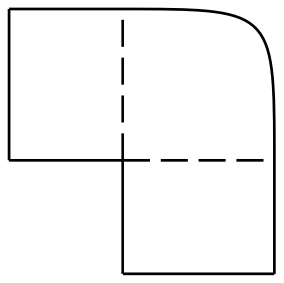

FIG. 3.

On the other hand, for any EIDD method the two data transfer operations in the first two steps are necessary for predicting the IBC and for implicitly computing the solution on the neighboring subdomain assigned to another processor. Thus compared with EIDD algorithms, the SEIDD algorithm has no extra communication cost, as the following theorem states.

THEOREM 3.1. *The stabilization* (*step* 3) *of SEIDD algorithm adds zero communication cost to EIDD algorithm.*

Furthermore, using any method (including the explicit forward Euler temporal discretization which requires only one matrix-vector multiplication for each time step), it needs solution data from nearby to compute the solution on the interface boundary. This incurs one data transferring operation. And to compute the solution near the interface boundary also incurs one data transferring operation. Thus time-stepwisely, the minimal number of data transferring operations is two for *any* parallel algorithm using *any* temporal discretization scheme. Therefore, we have arrived at the following conclusion concerning the communication cost of the parallel SEIDD methods.

THEOREM 3.2. *The parallel SEIDD algorithm is optimal in terms of number of data transferring operations for each time step.*

To carry out a quantitative analysis of the computation and communication efficiency of the parallel SEIDD algorithm, we assume that each of the two data transferring operations are carried out by $p - 1$ processors simultaneously with almost equal load. This can be easily achieved when the domain is decomposed as in Figures 2 or 3, where the interface boundaries do not cross into each other inside the problem domain. It is clear that for a given parallel machine size, namely, fixed p, the total communication cost per time step is an increasing function of the total "length" of the interface boundaries. Thus a domain partitioning strategy should be chosen to keep the total interface boundaries as "short" as possible as long as it does not incur excessive complication or deterioration of the algorithm. As shown in Figure 4, both partitioning strategies result in four subdomains of equal size, but strategy (a) produces "shorter" interface boundaries—only two-thirds of that resulting from strategy (b). However, with strategy (a), both the explicit prediction and the implicit stabilization of the interface boundary need special treatment to achieve good stability without increasing computation and communication costs. However, in this paper, we restrict the scope of discussion to the stabilization techniques under the case of no-crossover interface boundaries, and leave to a future project the interface boundary treatment for more general domain partitioning strategies that include crossover of interface boundaries.

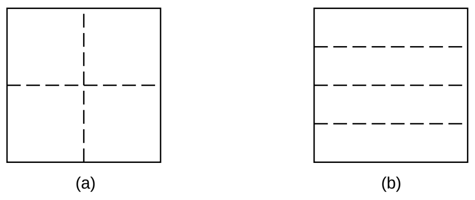

FIG. 4.

Now let N_Γ denote the number of grid points on the interface boundaries; then the total communication time of each time step satisfies

$$T_{\text{comm}} \leq 2\alpha \frac{N_\Gamma}{p-1} + \beta, \tag{3.2}$$

where α is some system-dependent data transfer rate and β is the communication startup overhead.

The computation cost of the stabilization process of the SEIDD algorithm is proportional to the number of grid points N_Γ on the interface boundary Γ_h. But since N_Γ is much smaller than the total number of grid points (denoted by N), the computation overhead of the stabilization is very small and negligible. Assume the function of computation cost bound for the predictor, the subdomain solver, and the stabilizer are the same and denoted by ϕ. Then the total computation cost at each time step is the sum of the prediction time $\phi(\frac{N_\Gamma}{p-1})$, the subdomain solver time $\phi(\frac{N-N_\Gamma}{p})$, and the stabilization time $\phi(\frac{N_\Gamma}{p-1})$, yielding

$$T_{\text{comp}} = 2\phi\left(\tfrac{N_\Gamma}{p-1}\right) + \phi\left(\tfrac{N-N_\Gamma}{p}\right). \tag{3.3}$$

Cost function usually increases faster than linearly or at least linearly, which means

$$\begin{cases} \phi(n) \geq p\,\phi\left(\tfrac{n}{p}\right), \\ \phi(n) \geq \phi(n-n') + \phi(n') \end{cases} \tag{3.4}$$

for $n' \leq n$. Then the parallel speedup, defined as single processor execution time T_1 over parallel execution time T_p, can be estimated from (3.2), (3.3), and (3.4) as follows:

$$\begin{aligned} S_p = T_1/T_p &= T_1/(T_{\text{comp}} + T_{\text{comm}}) = \frac{\phi(N)}{2\phi(\frac{N_\Gamma}{p-1}) + \phi(\frac{N-N_\Gamma}{p}) + 2\alpha\frac{N_\Gamma}{p-1} + \beta} \\ &\geq \frac{\phi(N)}{\frac{2\phi(N_\Gamma)}{p-1} + \frac{\phi(N)-\phi(N_\Gamma)}{p} + 2\alpha\frac{N_\Gamma}{p-1} + \beta} \approx \frac{p\,\phi(N)}{\phi(N) + \frac{p}{p-1}[2\alpha N_\Gamma + \beta + \phi(N_\Gamma)]}. \end{aligned}$$

And the corresponding efficiency, defined as speedup over the number of processors, is

$$E_p = \frac{\phi(N)}{\phi(N) + \frac{p}{p-1}[2\alpha N_\Gamma + \beta + \phi(N_\Gamma)]}.$$

As analyzed above, the SEIDD methods are computationally and communicationally efficient for *each time step*. If they further have good stability to ease the excessively small time step size restriction, the time-stepwisely efficient SEIDD will possess great potential for large simulation problems on distributed memory architecture machines.

4. The stabilized forward-backward Euler method. The SEIDD algorithm given in section 3 allows many choices for the explicit predictor in step 1, the subdomain scheme in step 2, and the stabilizer in step 3. Starting from this section, we shall present three SEIDD algorithms with different choices for the predictor, the subdomain temporal scheme, and the stabilizer.

Before presenting the algorithms, we introduce some notations. Let $L^2(\Omega_h)$ denote the function space defined on the finite-dimensional discrete space Ω_h equipped with the L^2 norm (same as the Euclid norm in this case). Let u_b^n and f^n denote the exterior boundary condition and the term $f(t,x)$ in (1.1) at time step n. For a subset $S \subset \Omega_h$, let χ_S be an operator on $L^2(\Omega_h)$ given by

$$\chi_S\, v(x) = \begin{cases} v(x), & x \in S, \\ 0, & x \notin S, \end{cases}$$

i.e., χ_S is a diagonal matrix with 1 on the positions corresponding to the grid points in the subset $S \subset \Omega_h$ and 0 elsewhere. We use I to denote the identity matrix on $L^2(\Omega_h)$. With these notations we are ready to present the first SEIDD algorithm, named SEIDD1.

We choose the forward Euler scheme for the first step of the SEIDD algorithm. Then the interface boundary condition predictor is mathematically representable as

$$\text{(4.1)} \qquad \begin{cases} \chi_{\Gamma_h} \frac{u_h^{n+1/3} - u_h^n}{\Delta t} = \chi_{\Gamma_h} \left(A_h u_h^n + A_b u_b^n + f^n\right), \\ \chi_{\Gamma_h^c}\, u_h^{n+1/3} = \chi_{\Gamma_h^c}\, u_h^n, \end{cases}$$

where A_b denotes the matrix that operates on discrete exterior boundary conditions. To discretize (1.1) on the subdomains, we choose the backward Euler scheme and obtain

$$\text{(4.2)} \qquad \begin{cases} \chi_{\Gamma_h^c} \frac{u_h^{n+2/3} - u_h^n}{\Delta t} = \chi_{\Gamma_h^c} \left(A_h u_h^{n+2/3} + A_b u_b^{n+1} + f^{n+1}\right), \\ \chi_{\Gamma_h}\, u_h^{n+2/3} = \chi_{\Gamma_h}\, u_h^{n+1/3}. \end{cases}$$

In the stabilization process the predicted interface boundary condition $\chi_{\Gamma_h} u_h^{n+1/3}$ is thrown away and $\chi_{\Gamma_h} u_h^n$ is brought back. Then the backward Euler scheme is used for the stabilization, so it has the representation

$$\text{(4.3)} \qquad \begin{cases} \chi_{\Gamma_h} \frac{u_h^{n+1} - u_h^n}{\Delta t} = \chi_{\Gamma_h} \left(A_h u_h^{n+1} + A_b u_b^{n+1} + f^{n+1}\right), \\ \chi_{\Gamma_h^c}\, u_h^{n+1} = \chi_{\Gamma_h^c}\, u_h^{n+2/3}. \end{cases}$$

Through tedious but simple calculations, the domain decomposition method (4.1)–(4.3) can be written into

$$\text{(4.4)} \qquad \begin{aligned} u_h^{n+1} &= (I - \Delta t A_1)^{-1} \left[\chi_{\Gamma_h} + \chi_{\Gamma_h^c} (I - \Delta t A_2)^{-1} (I + \Delta t A_1)\right] u^n \\ &\quad + \Delta t (I - \Delta t A_1)^{-1} \left[\chi_{\Gamma_h} g^{n+1} + \chi_{\Gamma_h^c} (I - \Delta t A_2)^{-1} (\chi_{\Gamma_h^c} g^{n+1} + \chi_{\Gamma_h} g^n)\right], \end{aligned}$$

where $A_1 = \chi_{\Gamma_h} A_h$, $A_2 = \chi_{\Gamma_h^c} A_h$, and $g^k = A_b u_b^k + f^k$. For homogeneous problems where $u_b = 0$ and $f = 0$, it is easily obtainable from (4.4) that

$$\text{(4.5)} \qquad u_h^n = (I - \Delta t A_1)^{-1} \widehat{G}(\Delta t, h)_n (I - \Delta t A_1) u_h^0,$$

where

$$\widehat{G}(\Delta t, h) = \left[\chi_{\Gamma_h} + \chi_{\Gamma_h^c} (I - \Delta t A_2)^{-1}(I + \Delta t A_1)\right] (I - \Delta t A_1)^{-1}. \tag{4.6}$$

Now we can see that the stability of SEIDD1 depends very much on $\widehat{G}$. The main instability-causing term in SEIDD is the explicit predictor $I+\Delta t A_1$, which, as can be seen from (4.6), is likely to be somewhat stabilized from right by the term $(I - \Delta t A_1)^{-1}$—the term corresponding to the stabilizer in the previous time step with respect to the time step of the predictor. Therefore, a better stability condition for SEIDD is expected than for EIDD.

5. The stabilized Dawson–Du–Dupont method. In this section we present a stabilizer for the explicit-implicit algorithm of Dawson, Du, and Dupont. We call the stabilized algorithm the SEIDD2 algorithm. The emphasis of this section is on illustrating how to design a stabilizer for a given interface boundary condition predictor and compare the difference of stabilization with implicit correction.

Let A_h^x denote the x-directional difference component of the operator A_h, and A_h^y the y-directional difference component. Let A_b^x be the x-direction boundary condition matrix that operates on the exterior boundary condition, and let A_b^y the y-direction boundary condition matrix. Then we represent the two steps of Dawson–Du–Dupont algorithm by

$$\begin{cases} \chi_{\Gamma_h} \frac{u_h^{n+1/3} - u_h^n}{\Delta t} = \chi_{\Gamma_h} \left(A_h^x u_h^n + A_h^y u_h^{n+1/3} + A_b^x u_b^n + A_b^y u_b^{n+1} + f^n\right), \\ \chi_{\Gamma_h^c} u_h^{n+1/3} = \chi_{\Gamma_h^c} u_h^n \end{cases} \tag{5.1}$$

and

$$\begin{cases} \chi_{\Gamma_h^c} \frac{u_h^{n+2/3} - u_h^n}{\Delta t} = \chi_{\Gamma_h^c} \left(A_h u_h^{n+2/3} + A_b u_b^{n+1} + f^{n+1}\right), \\ \chi_{\Gamma_h} u_h^{n+2/3} = \chi_{\Gamma_h} u_h^{n+1/3}. \end{cases} \tag{5.2}$$

To stabilize the predictor (5.1), we start from the backward Euler method

$$\begin{cases} \chi_{\Gamma_h} \frac{u_h^{n+1} - u_h^n}{\Delta t} = \chi_{\Gamma_h} \left(A_h u_h^{n+1} + A_b u_b^{n+1} + f^{n+1}\right), \\ \chi_{\Gamma_h^c} u_h^{n+1} = \chi_{\Gamma_h^c} u_h^{n+2/3} \end{cases} \tag{5.3}$$

for the spatially discretized problem (3.1) on the interface boundary. We rewrite the first equation in (5.3) into

$$\chi_{\Gamma_h} (I - \Delta t A_h) u_h^{n+1} = \chi_{\Gamma_h} \left[u_h^n + \Delta t(A_b u_b^{n+1} + f^{n+1})\right].$$

Since $A_h = A_h^x + A_h^y$, an approximate factorization of the left-hand side turns the above equation together with the second equation in (5.3) into

$$\begin{cases} \chi_{\Gamma_h}(I - \Delta t A_1^y)(I - \Delta t A_1^x) u_h^{n+1} = \chi_{\Gamma_h}\left[u_h^n + \Delta t(A_b u_b^{n+1} - A_b^y A_b^x u_b^{n+1} + f^{n+1})\right], \\ \chi_{\Gamma_h^c} u_h^{n+1} = \chi_{\Gamma_h^c} u_h^{n+2/3}, \end{cases} \tag{5.4}$$

where $A_1^x = \chi_{\Gamma_h} A_h^x$ and $A_1^y = \chi_{\Gamma_h} A_h^y$.

In the stabilization step (5.4), we place the operator $(I - \Delta t A_h^y)$ to the left of $(I - \Delta t A_h^x)$. This ordering is important for the stabilization. The importance of this ordering can be seen from an analysis given below.

For the homogeneous problem where the boundary conditions $u_b = 0$ and $f = 0$, the SEIDD2 method (5.1), (5.2), (5.4) can be written into

$$u^{n+1} = S(\Delta t, h) \left[\chi_{\Gamma_h} + \chi_{\Gamma_h^c} (I - \Delta t A_2)^{-1} P(\Delta t, h) \right] u^n, \tag{5.5}$$

where $S(\Delta t, h)$ is the stabilizer given by

$$S(\Delta t, h) = (I - \Delta t A_1^x)^{-1} (I - \Delta t A_1^y)^{-1}, \tag{5.6}$$

$P(\Delta t, h)$ is the interface boundary condition predictor $P(\Delta t, h) = (I - \Delta t A_1^y)^{-1}(I + \Delta t A_1^x)$, and $A_2 = \chi_{\Gamma_h^c} A_h$. It follows from (5.5) that

$$u^n = S(\Delta t, h) \widehat{G}(\Delta t, h)^n S(\Delta t, h)^{-1} u^0, \tag{5.7}$$

where $\widehat{G}(\Delta t, h) = [\chi_{\Gamma_h} + \chi_{\Gamma_h^c} (I - \Delta t A_2)^{-1} P(\Delta t, h)] S(\Delta t, h)$. From (5.7) we can see that the convergence of the SEIDD2 method depends on the stability of $\widehat{G}(\Delta t, h)$, which in turn depends on the stability of $P(\Delta t, h)S(\Delta t, h)$. From the equation

$$P(\Delta t, h)S(\Delta t, h) = [(I - \Delta t A_1^y)^{-1}(I + \Delta t A_1^x)] \cdot [(I - \Delta t A_1^x)^{-1}(I - \Delta t A_1^y)^{-1}],$$

we can see that $(I - \Delta t A_1^x)^{-1}$ in $S(\Delta t, h)$ might be able to stabilize the explicit component $(I + \Delta t A_1^x)$ in $P(\Delta t, h)$. However, if the ordering of the two operators in $S(\Delta t, h)$ is reversed, namely,

$$S(\Delta t, h) = (I - \Delta t A_1^y)^{-1} (I - \Delta t A_1^x)^{-1}, \tag{5.8}$$

then $P(\Delta t, h)S(\Delta t, h) = \left[(I - \Delta t A_1^y)^{-1}(I + \Delta t A_1^x)\right] \cdot \left[(I - \Delta t A_h^y)^{-1}(I - \Delta t A_h^x)^{-1}\right]$, which may not necessarily produce good stability since the operator $(I - \Delta t A_h^y)^{-1}$ in $S(\Delta t, h)$ does not seem to be able to stabilize the explicit part $(I + \Delta t A_1^x)$ of $P(\Delta t, h)$ in all situations.

The design of the stabilizer for Dawson, Du, and Dupont's EIDD algorithm is a good illustration of the difference between stabilization and implicit correction mentioned in section 2.5. Implicit correction considers mainly the errors of the interface boundary conditions, while stabilization also concerns the errors caused by the prediction that have propagated to the subdomains. Thus, the focus of stabilization is to stabilize the explicit prediction (not just the errors on the interface boundaries caused by the prediction) so that the prediction-caused interface boundary errors that propagate to the subdomains could have already been stabilized by the stabilizer of the previous time step. Therefore, for implicit correction, the factorized implicit scheme (5.8) and the backward Euler scheme could both be feasible choices since they can reduce the errors on the interface boundaries to an acceptable range. But the stabilization considers only the factorized scheme (5.6), which is particularly ordered according to the predictor.

6. A method with a factorized subdomain scheme.

Both the SEIDD1 and SEIDD2 algorithms require solving an elliptic equation of the form

$$(I - \Delta t A) u = r \tag{6.1}$$

on the subdomains. When the spatial operator A is not separable or even nonsymmetric, iterative solvers must be employed to solve the elliptic equation for each time step. In this section, we propose an algorithm which factorizes the left-hand side of

(6.1) with a direction-based splitting of the spatial operator A. This factorization reduces the computation cost to a linear order of $O(N)$ for a total of N grid points on the subdomains. With this factorization, the factorized SEIDD algorithm becomes completely noniterative, both globally and on each subdomain.

We choose to present the factorization for the SEIDD1 algorithm. The factorization for the SEIDD2 is exactly the same. We assume that the spatial domain of the parabolic equation is two-dimensional and the spatial operator A is splittable as $A = A^x + A^y$. For example, when A is the two-dimensional Laplace operator $A = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, $A^x = \frac{\partial^2}{\partial x^2}$ and $A^y = \frac{\partial^2}{\partial y^2}$ form a directional splitting of the operator A. The SEIDD algorithm with a directionally factorized subdomain temporal scheme is given below:

$$\begin{cases} \chi_{\Gamma_h} u_h^{n+1/3} = \chi_{\Gamma_h} [(I + \Delta t A_h) u_h^n + A_b u_b^n + \Delta t f^n], \\ \chi_{\Gamma_h^c} u_h^{n+1/3} = \chi_{\Gamma_h^c} u_h^n, \end{cases} \tag{6.2}$$

$$\begin{cases} \chi_{\Gamma_h^c} (I - \Delta t A_h^x)(I - \Delta t A_h^y) u_h^{n+2/3} = \chi_{\Gamma_h^c} [u_h^n + \Delta t (A_b - \Delta t A_b^x A_b^y) u_b^{n+1} + \Delta t f^{n+1}], \\ \chi_{\Gamma_h} u_h^{n+2/3} = \chi_{\Gamma_h} u_h^{n+1/3}, \end{cases} \tag{6.3}$$

and

$$\begin{cases} \chi_{\Gamma_h} (I - \Delta t A_h)(u_h^{n+1} + u_b^{n+1}) = \chi_{\Gamma_h} [u_h^n + \Delta t f^{n+1}], \\ \chi_{\Gamma_h^c} u_h^{n+1} = \chi_{\Gamma_h^c} u_h^{n+2/3}. \end{cases} \tag{6.4}$$

We call the algorithm (6.2)–(6.4) the SEIDD3 algorithm. The difference between this algorithm and SEIDD1 is that in the second step (6.3), the SEIDD3 algorithm has $(I - \Delta t A_h^x)(I - \Delta t A_h^y)$ on the left-hand side while the SEIDD1 algorithm has $(I - \Delta t A_h)$. For a two-dimensional problem, the discretized directional components of A_h are usually tridiagonal matrices [2]. Thus $(I - \Delta t A_h^x)$ and $(I - \Delta t A_h^y)$ can be easily inverted with a computation cost of linear order. On the other hand, the factorization introduces an $O((\Delta t)^2)$ error. Since $A = A^x + A^y$, we have that

$$(I - \Delta t A_h^x)(I - \Delta t A_h^y) = I - \Delta t A_h + (\Delta t)^2 A_h^x A_h^y.$$

Thus the error introduced by factorizing $(I - \Delta t A_h)$ into $(I - \Delta t A_h^x)(I - \Delta t A_h^y)$ is $(\Delta t)^2 A_h^x A_h^y$. But the predictor, the subdomain scheme, and the stabilizer all have $O\left((\Delta t)^2\right)$ temporal truncation errors and the SEIDD1 algorithm is also first order (see [39]). Thus the error introduced by the directional factorization of the subdomain scheme does not decrease the order of temporal accuracy.

7. Experimental results. In this section, we present some numerical results obtained by applying the SEIDD algorithms to parabolic problems. Four different types of problems have been chosen to test the proposed domain decomposition algorithms: a problem with a symmetric and negative definite spatial operator, a problem with a symmetric and indefinite spatial operator, several problems with nonsymmetric spatial operators, and an unstable problem with a nonsymmetric and indefinite spatial operator. In the experiments, stability is carefully examined together with parallel speedup and efficiency of the SEIDD algorithms.

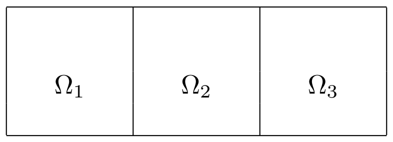

FIG. 5.

7.1. Stability testing.

7.1.1. The heat equation. We apply the SEIDD algorithms to the two-dimensional heat equation

$$\begin{cases} \frac{\partial u(t,x,y)}{\partial t} = (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})u, & x \in \Omega, \quad t \geq 0 \\ u(t,x,y) = u_b(t,x,y), & x \in \partial\Omega, \quad t \geq 0, \\ u(0,x,y) = u^0(x,y), & x \in \Omega, \end{cases} \tag{7.1}$$

on the rectangular domain $\Omega = [0,3] \times [0,1]$. The heat equation has a "good" spatial operator—the Laplacian, which is symmetric and negative definite. In the numerical experiment, we partition the domain into 192 intervals of length $h = 1/64$ in the x-dimension, and partition the y-dimension into 64 intervals of the same mesh size h as in x-dimension. On this regularly structured grid, we use the second order central finite difference

$$\frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} = f_i'' \tag{7.2}$$

for the discretization of the Laplace operator with the discrete Laplace operator denoted by Δ_h. Then the spatially discretized heat equation has the form $\frac{d}{dt}u_h = \Delta_h u_h + b_h(t)$, where $b_h(t)$ is resulted from the boundary conditions.

We solved the spatially discretized heat equation using the three SEIDD algorithms presented in sections 4 through 6, with the domain divided into three equal-sized squares as in Figure 5. The time intervals chosen for simulation are the unit interval $[0,1]$. We have used several different time discretization sizes Δt. The experiments were carried out on an NCSA Origin 2000 machine using 3 processors, each of 250 MHz, running an IRIX 6.5.9 operating system. 64-bit arithmetic operations were used in the numerical experiments. The measured errors of numerical solutions at time $t = 1$ are listed in Table 1 for the indicated temporal discretization sizes.

Since the SEIDD algorithms are stabilized EIDD algorithms, for stability comparison we also solved the heat equation using the EIDD algorithms—the SEIDD algorithms without the stabilization. On the other hand, since the backward Euler (listed as BEuler in the tables) method is the most stable method due to Widder's theorem [1], it can be considered the benchmark for stability. We solved the heat equation using the backward Euler method on the entire nonpartitioned domain by one processor, and the measured errors of the solutions computed by the BEuler method are also listed in Table 1. It is well known that for an unconditionally stable method, the simulation error remains small even when the time step size Δt is large relative to the spatial mesh size [30]. As indicated by the experimental results in Table 1, the errors of the SEIDD algorithms remain relatively small when the time step size Δt is large, and they are almost as small as those of the backward Euler method, experimentally supporting the effectiveness of the stabilization of the SEIDD algorithms.

Table 1

$u_t = \Delta u$ with $u(t,x,y) = e^{-2t}\cos(x+y)$. The table lists the maximal error of the solution at $t = 1$. The symbol ∞ denotes an error larger than $1.0e + 100$. The spatial domain is $[0,3]\times[0,1]$ with a mesh size $h = 1/64$.

Δt	1/50	1/100	1/200	1/400	1/800
SEIDD1	4.2e–03	8.9e–04	1.5e–04	5.3e–05	3.1e–05
SEIDD2	3.8e–04	2.2e–04	1.2e–04	6.8e–05	3.6e–05
SEIDD3	4.8e–03	1.1e–03	2.2e–04	3.7e–05	1.4e–05
BEuler	6.1e–04	3.0e–04	1.5e–04	7.6e–05	3.8e–05
EIDD1	∞	∞	∞	∞	∞
EIDD2	3.9e+54	2.4e+98	∞	∞	∞
EIDD3	∞	∞	∞	∞	∞

7.1.2. An unstable diffusion problem. In this section, we test the SEIDD algorithms on the problem

$$\begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + 3u, & x\in\Omega, \quad t\geq 0 \\ u(t,x,y) = 0, & x\in\partial\Omega, \quad t\geq 0, \\ u(0,x,y) = u^0(x,y), & x\in\Omega, \end{cases} \tag{7.3}$$

on the rectangular domain $\Omega = [0,2\pi]\times[0,\pi]$. On this spatial domain, the eigenvalues of the Laplace operator are $-\frac{j^2}{4} - k^2$ for $j,k = 1,2,3,\ldots$. Thus the spatial operator $A = -\Delta + 3$ has positive eigenvalues and the homogeneous problem (7.3) could have solutions of exponential growth, e.g., $u(t,x,y) = e^t \sin(x)\sin(y)$ is a solution for the initial condition $u(0,x,y) = \sin(x)\sin(y)$. An evolutionary (or dynamical) system is stable if and only if for all initial conditions and bounded boundary conditions its solutions stay uniformly bounded as time progresses. This necessitates that all eigenvalues of the spatial operator be nonpositive for a stable system. Hence the testing problem (7.3) is unstable.

In the numerical experiment, we choose a spatial grid with the x-dimension divided into 256 subintervals of equal length $h = \pi/128$ and the y-dimension into 128 intervals of the same mesh size $h = \pi/128$. On this regularly structured grid, we use the second order central finite difference (7.2) for the discretization of the Laplace operator and obtain the following spatially discretized problem:

$$\frac{du_h}{dt} = (\Delta_h + 3)u_h. \tag{7.4}$$

We solved the spatially discretized diffusion equation by the SEIDD algorithms with the domain divided into two equal-sized square subdomains by line $x = \pi$. The time interval chosen for simulation is the unit interval $[0,1]$. Several different time discretization sizes Δt have been used. The measured errors of numerical solutions at time $t = 1$ are listed in Table 2 for the indicated temporal discretization sizes.

To examine the stabilization effectiveness of SEIDD algorithms, we also have solved the unstable diffusion problem using the EIDD algorithms and listed the solution errors in Table 2. For stability comparison, we solved the discrete problem (7.4) using the backward Euler method on the entire nonpartitioned domain by one processor. Measured errors of the solutions computed by the backward Euler method (BEuler) are also listed in Table 2. As indicated by the experimental results in Table 2, the errors of the SEIDD1 and SEIDD2 algorithms remain relatively small when the time step size Δt is large, and they are the same as the errors of the backward

TABLE 2

$u_t = \Delta u + 3u$ with $u(t,x,y) = e^t \sin(x)\sin(y)$. The spatial domain is $[0,2\pi]\times[0,\pi]$ with mesh size $h = \pi/128$.

Δt	1/25	1/50	1/100	1/200	1/400	1/800
SEIDD1	5.7e–02	2.8e–02	1.4e–02	7.1e–03	3.7e–03	2.0e–03
SEIDD2	5.7e–02	2.8e–02	1.4e–02	7.1e–03	3.7e–03	2.0e–03
SEIDD3	3.0e–01	1.4e–01	7.0e–02	3.5e–02	1.7e–02	8.8e–03
BEuler	5.7e–02	2.8e–02	1.4e–02	7.1e–03	3.7e–03	2.0e–03
EIDD1	3.5e+45	4.4e+94	∞	∞	∞	∞
EIDD2	3.6e+12	4.3e+33	4.3e+65	∞	∞	∞
EIDD3	2.6e+42	5.9e+88	∞	∞	∞	∞

Euler method. The errors of the SEIDD3 algorithm are about 3 to 5 times larger than those of the backward Euler but are much smaller than those of the EIDD algorithms.

7.1.3. Convection-diffusion problems. In this section, we test the SEIDD algorithms on the problem

$$(7.5)\qquad \begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + 9.9\sin(x)\frac{\partial}{\partial x}u - 9.9\cos(x)u, & x\in\Omega, \quad t\geq 0,\\ u(t,x,y) = 0, & x\in\partial\Omega, \quad t\geq 0,\\ u(0,x,y) = u^0(x,y), & x\in\Omega, \end{cases}$$

on the rectangular domain $\Omega = [0,2\pi]\times[0,\pi]$. In addition to being nonsymmetric, the problem is also unstable, since the twice differentiable function $f(x,y) = \sin(x/2)\sin(y)$ is an eigenvector of A with a positive eigenvalue 8.65.

We partition the x-dimension of the domain into 256 intervals of length $h = \pi/128$ and partition the y-dimension into 128 intervals of the same mesh size $h = \pi/128$. On this grid we use the second order finite difference (7.2) for the discretization of the Laplace operator and discretize $\frac{\partial}{\partial x}u$ by the central finite difference

$$(7.6)\qquad \frac{f_{i+1} - f_{i-1}}{h} = f'_i.$$

We solved the spatially discretized equation by the SEIDD algorithms with the domain divided into two equal-sized squares by line $x = \pi$. The time interval chosen for simulation is the unit interval $[0,1]$, and several different time discretization sizes Δt were used. The measured errors of numerical solutions at time $t = 1$ are listed in Table 3 for the indicated temporal discretization sizes. We have also solved the problem using the EIDD algorithms and the backward Euler method. The solution errors computed by the EIDD and backward Euler algorithms are listed in Table 3.

As indicated by the experimental results in Table 3, the errors of the SEIDD1 and SEIDD2 algorithms remain small even when the time step size Δt is large, and the errors are the same as those of the backward Euler method. The errors of the SEIDD3 algorithm are about 30 to 100 times larger than the backward Euler but are still much smaller than the errors computed by the EIDD methods.

To see how different degrees of nonsymmetry and instability of the problem affect the numerical error or stability of the SEIDD3 algorithm, we test the SEIDD3 algorithm on the problem

$$\begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + \alpha\sin(x)\frac{\partial}{\partial x}u - \alpha\cos(x)u, & x\in\Omega, \quad t\geq 0,\\ u(t,x,y) = 0, & x\in\partial\Omega, \quad t\geq 0,\\ u(0,x,y) = u^0(x,y), & x\in\Omega, \end{cases}$$

Table 3

$u_t = \Delta u + 9.9\sin(x)u_x - 9.9\cos(x)u$ with $u = e^{-2t}\sin(x)\sin(y)$. The spatial domain is $[0, 2\pi]\times[0, \pi]$ with mesh size $h = \pi/128$

Δt	1/25	1/50	1/100	1/200	1/400	1/800
SEIDD1	1.1e–02	5.5e–03	2.8e–03	1.5e–03	8.0e–04	4.6e–04
SEIDD2	1.1e–02	5.5e–03	2.8e–03	1.5e–03	8.0e–04	4.6e–04
SEIDD3	1.9e+00	3.7e–01	1.3e–01	5.4e–02	2.5e–02	1.2e–02
BEuler	1.1e–02	5.5e–03	2.8e–03	1.5e–03	8.0e–04	4.6e–04
EIDD1	8.0e+42	1.8e+88	∞	∞	∞	∞
EIDD2	4.0e+13	1.9e+34	2.0e+66	∞	∞	∞
EIDD3	1.7e+42	4.4e+88	∞	∞	∞	∞

Table 4

$u_t = \Delta u + \alpha\sin(x)u_x - \alpha\cos(x)u$ with $u = e^{t}\sin(x)\sin(y)$. The spatial domain is $[0, 2\pi]\times[0, \pi]$ with mesh size $h = \pi/128$.

α	Δt	1/25	1/50	1/100	1/200	1/400	1/800
0.9	SEIDD3	6.4e–03	3.2e–03	1.6e–03	8.2e–04	4.2e–04	2.2e–04
3.9	SEIDD3	4.5e–02	2.1e–02	1.0e–02	5.2e–03	2.6e–03	1.3e–03
6.9	SEIDD3	2.8e–01	1.0e–01	4.4e–02	2.1e–02	1.0e–02	5.0e–03
9.9	SEIDD3	1.9e+00	3.7e–01	1.3e–01	5.4e–02	2.5e–02	1.2e–02
0.9	BEuler	1.1e–02	5.4e–03	2.7e–03	1.4e–03	6.9e–04	3.5e–04
3.9	BEuler	1.1e–02	5.4e–03	2.7e–03	1.4e–03	7.2e–04	3.8e–04
6.9	BEuler	1.1e–02	5.5e–03	2.8e–03	1.4e–03	7.6e–04	4.2e–04
9.9	BEuler	1.1e–02	5.5e–03	2.8e–03	1.5e–03	8.0e–04	4.6e–04

with varying α. The testing results are listed in Table 4.

We also solved the same problem with varying coefficients α by the backward Euler method. The maximal errors computed by the backward Euler method are listed in Table 4. The measured numerical errors indicate that when α is small, the SEIDD3 algorithm exhibits quite small errors. As α increases, the numerical errors also increase.

7.1.4. An unstable convection-diffusion problem. In this section, we test the SEIDD algorithms on the problem

$$(7.7)\quad \begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + \sin(x)\frac{\partial}{\partial x}(\sin(x)u) + (3-\sin(2x))u, & x\in\Omega, \quad t\geq 0,\\ u(t,x,y) = 0, & x\in\partial\Omega, \quad t\geq 0,\\ u(0,x,y) = u^0(x,y), & x\in\Omega, \end{cases}$$

on the rectangular domain $\Omega = [0, 2\pi]\times[0, \pi]$. On $L_0^2(\Omega)$, the spatial operator

$$Au = \Delta u + \sin(x)\tfrac{\partial}{\partial x}(\sin(x)u) + (3 - \sin(2x))u$$

is indefinite, since, as proven in [39], the function $v(x, y) = \sin(0.5x)\sin(y) \in L_0^2(\Omega)$ satisfies $\langle Av, v\rangle + \langle v, Av\rangle \geq 1.5\|v\|^2$.

To solve the problem numerically, we partition the x-dimension of the domain into 256 intervals of length $h = \pi/128$ and partition the y-dimension into 128 intervals of the same mesh size $h = \pi/128$. On this grid, we use the second order finite difference (7.2) for the discretization of the Laplace operator and discretize $\frac{\partial}{\partial x}u$ by the central finite difference (7.6). We solved the spatially discretized equation by the SEIDD algorithms with the domain divided into two equal-sized squares by line $x = \pi$. The time interval chosen for simulation is the unit interval $[0, 1]$. We used several different

TABLE 5
$u_t = \Delta u + \sin(x)^2 u_x + [3 - \sin(x)\cos(x)]u$ with $u = e^t \sin(x)\sin(y)$. The spatial domain is $[0, 2\pi] \times [0, \pi]$ with mesh size $h = \pi/128$.

Δt	1/25	1/50	1/100	1/200	1/400	1/800
SEIDD1	5.7e–02	2.8e–02	1.4e–02	7.1e–03	3.7e–03	2.0e–03
SEIDD2	5.7e–02	2.8e–02	1.4e–02	7.2e–03	3.7e–03	2.0e–03
SEIDD3	3.9e–01	1.9e–01	9.1e–02	4.5e–02	2.3e–02	1.1e–02
BEuler	5.7e–02	2.8e–02	1.4e–02	7.2e–03	3.7e–03	2.0e–03
EIDD1	4.4e+45	5.6e+94	∞	∞	∞	∞
EIDD2	8.6e+21	2.2e+42	1.6e+74	∞	∞	∞
EIDD3	2.6e+42	6.1e+88	∞	∞	∞	∞

time discretization sizes Δt. The measured errors of numerical solutions at time $t = 1$ are listed in Table 5 for the indicated temporal discretization sizes. We also have solved the problem using the EIDD algorithms and the backward Euler method. The solution errors computed by the EIDD and backward Euler algorithms are also listed in the Table 5.

As indicated by the experimental results in Table 5, the errors of the SEIDD1 and SEIDD2 algorithms remain small even when the time step size Δt is large, and the errors are almost the same as those of the backward Euler method. The errors of the SEIDD3 algorithm are about 5 times larger than the backward Euler but are still much smaller than the errors computed by the EIDD algorithms. The numerical experiment data show that the SEIDD1 and SEIDD2 algorithms are robust and exhibit good stability for unstable, nonselfadjoint problems.

7.1.5. The effects of the subdomain size. To see how the subdomain size affects the stability and numerical error of the method, we choose a convection-diffusion problem

$$\begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + 10\, u_x + 6\sin(0.6x + 1.1y), & x \in \Omega, \quad t \geq 0, \\ u(t,x,y) = u_b(t,x,y), & x \in \partial\Omega, \quad t \geq 0, \\ u(0,x,y) = u^0(x,y), & x \in \Omega, \end{cases} \tag{7.8}$$

on the unit square $[0,1] \times [0,1]$ with a uniform mesh $h_x = h_y = 1/64$. We divide the square domain into p subdomains of equal size by vertical lines as in Figure 6. We tested the SEIDD1 method on this problem with p ranging from 1 to 32, where 32 is the largest possible number for each subdomain to have nonempty interior, i.e., only one vertical line of interior grid points. The time interval chosen for simulation is $[0, 1]$. We used several different time discretization sizes Δt. The measured errors of the numerical solutions at time $t = 1$ are listed in Table 6 for the indicated subdomain width (W) and temporal discretization sizes. The width of the subdomains satisfies $W = 64h/p$, and hence $W = 64h$ corresponds to the backward Euler method. We also have solved the problem using the EIDD1 algorithm, and the errors of the numerical solutions computed by the EIDD1 algorithm are listed in the table.

The experimental data show that the SEIDD methods retain good stability for all tested cases, although the numerical error for simultaneously small W and large Δt are relatively large when compared to those from the backward Euler method. We further tested the SEIDD1 using one of the partitions in Table 6 that produces the narrowest subdomains, i.e., the partition where the unit square domain is divided into 32 equal-sized subdomains. In the tests, we choose large Δt's with respect to h's and let Δt shrink proportionally with h so that their ratios remain fixed. The maximal

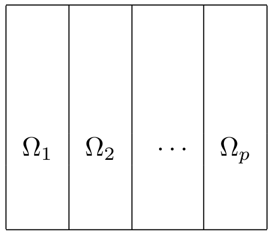

Fig. 6.

Table 6
$u_t = \Delta u + 10u_x + 6e^{-1.57t}\sin(0.6x+1.1y)$ with $u = e^{-1.57t}\cos(0.6x+1.1y)$. The spatial domain is $[0,1]\times[0,1]$ with mesh size $h = 1/64$.

W	Δt	1/25	1/50	1/100	1/200	1/400	1/800	1/1600
64h	SEIDD1	4.0e–04	2.0e-04	1.0e-04	5.0e-05	2.6e-05	1.3e-05	7.1e-06
32h	SEIDD1	6.9e–03	1.8e-03	3.8e-04	7.7e-05	1.0e-05	8.1e-06	5.8e-06
16h	SEIDD1	2.9e–02	2.7e-03	9.6e-04	2.1e-04	4.0e-05	3.2e-06	3.4e-06
8h	SEIDD1	6.8e–02	2.3e-02	1.9e-03	4.6e-04	1.0e-04	1.8e-05	8.5e-07
4h	SEIDD1	3.3e–01	7.2e-02	4.1e-03	9.6e-04	2.3e-04	5.0e-05	8.6e-06
2h	SEIDD1	5.1e–01	9.8e-02	2.9e-02	1.9e-03	4.8e-04	1.1e-04	2.4e-05
32h	EIDD1	4.0e+66	∞	∞	∞	∞	∞	∞
16h	EIDD1	4.1e+66	∞	∞	∞	∞	∞	∞
8h	EIDD1	4.2e+66	∞	∞	∞	∞	∞	∞
4h	EIDD1	4.0e+66	∞	∞	∞	∞	∞	∞
2h	EIDD1	2.6e+66	∞	∞	∞	∞	∞	∞

Table 7
$u_t = \Delta u + 10u_x + 6e^{-1.57t}\sin(0.6x+1.1y)$ with $u = e^{-1.57t}\cos(0.6x+1.1y.)$ The domain $[0,1]\times[0,1]$ is divided by 31 vertical lines into 32 equal-sized subdomains.

$\Delta t/h$	$h=$	1/64	1/128	1/256	1/512	1/1024
	$\Delta t=$	1/32	1/64	1/128	1/256	1/512
2	Error=	$4.0e-01$	$1.9e-01$	$1.6e-01$	$9.2e-02$	$2.4e-03$
	$\Delta t=$	1/50	1/100	1/200	1/400	1/800
1.28	Error=	$9.8e-02$	$1.2e-01$	$3.7e-02$	$7.0e-03$	$2.0e-03$
	$\Delta t=$	1/100	1/200	1/400	1/800	1/1600
0.64	Error=	$2.9e-02$	$5.6e-03$	$2.0e-03$	$9.9e-04$	$5.0e-04$

errors at time $t = 1$ are listed in Table 7. These measured errors seem to suggest that as long as the partition of the domain is fixed, the SEIDD1 method converges even when the subdomain width is small and the Δt is relatively large.

7.2. Efficiency and scalability testing. Analyses given in section 3 show that the stabilization of SEIDD methods add negligible costs (computation and communication) to EIDD methods when the subdomains have a much larger number of grid points than the interface boundaries. To experimentally examine the overhead of the stabilization, we carried out a numerical experiment to compare the total cost (computation and communication) of SEIDD versus EIDD. We chose the heat equation on spatial domain $[0,8\pi]\times[0,2\pi]$. The domain is divided into four square subdomains of equal size, each assigned to a processor. In the test, we chose a sufficiently small time step size so that all EIDD methods converged for the testing problem. The measured execution times are listed in Table 8. From the testing data, no increased computa-

TABLE 8

Execution Time: SEIDD methods vs. EIDD methods. The equation is $u_t = u_{xx} + u_{yy}$ with $u = e^{-2t}\cos(x+y)$. The domain is $[0, 8\pi]\times[0, 2\pi]$, which is divided into 4 subdomains. Spatial mesh size $h = \pi/64$ with each subdomain of 128×128 grid points. The simulation time interval is $[0, 1]$ with time step size of $\triangle t = 1/2000$.

Method	T_total	T_comp	T_comm	Max_Err	nprocs
SEIDD1	5.3e+01	5.24e+01	3.0e–01	2.1e–04	4
SEIDD2	5.4e+01	5.28e+01	3.0e–01	2.1e–04	4
SEIDD3	2.5e+01	2.44e+01	3.0e–01	1.3e–04	4
EIDD1	5.3e+01	5.22e+01	3.0e–01	2.1e–04	4
EIDD2	5.3e+01	5.25e+01	3.0e–01	2.1e–04	4
EIDD3	2.5e+01	2.46e+01	3.0e–01	1.3e–04	4

tion time beyond machine variation range was recorded, experimentally supporting the analysis that SEIDD methods incur negligible amount of overhead by stabilizing EIDD methods.

To examine the scalability of the SEIDD algorithms, we apply the SEIDD1 and SEIDD3 to the convection-diffusion equation

$$\begin{cases} \frac{\partial u(t,x,y)}{\partial t} = \Delta u + \sin(x)u_x - \cos(x)u, & x\in\Omega, \quad t\geq 0, \\ u(t,x,y) = 0, & x\in\partial\Omega, \quad t\geq 0, \\ u(0,x,y) = u^0(x,y), & x\in\Omega, \end{cases}$$

on spatial domains $[0, p\pi]\times[0, \pi]$, where p is the number of subdomains. The domain partitioning is along the x-direction as shown in Figure 2, with each subdomain being a square and assigned to a processor. Uniform grid is applied to the domain with mesh size $\frac{\pi}{256}$ in both the x- and y-directions. On this grid we use the second order finite difference (7.2) for the discretization of the Laplace operator and discretize $\frac{\partial}{\partial x}u$ by the central finite difference (7.6). The simulation time interval is $[0, 1]$, and the time step size is $\Delta t = 1/200$.

We solved the problem by the SEIDD1 and SEIDD3 algorithms on a dedicated queue of an NCSA Origin 2000 machine with a maximum of 256 nodes, each of 250 MHz, running an IRIX 6.5.9 operating system. In the experiments, we measured the computation time (T_comp), the communication time (T_comm), the total execution time (T_total), and the maximal errors of the numerical solutions at time $t = 1$. These measured data are listed in Tables 9 and 10 together with parallel speedup and efficiency calculated using the total execution time T_total. In the tables, the unit of T_comp, T_comm, and T_total is second.

The experimental data show that the computation time almost remains the same as the problem size increases with the machine ensemble size such that the memory usage on each processor remains the same, namely, the problem size is scaled up following the memory-bounded constraint [35]. This phenomenon is well under expectation since the stabilization has a very low computation overhead. The communication time increases slowly as the number of processors increases. As the number of processors increases from 1 to 128, the efficiency has decreased less than 4 percentage points for the SEIDD1 algorithm and dropped about 9 percentage points for the SEIDD3 algorithm. This rate of performance decrease is very slow, matching well with the analysis given in section 3.

The difference in the decreased percentage between the SEIDD1 and SEIDD3 algorithms is due to the difference of computation costs of the two algorithms. The computation time used by the SEIDD3 algorithm is less than half of that used by the

Table 9

Solving $u_t = \Delta u + \sin(x)u_x - \cos(x)u$ by SEIDD1. Speedup and efficiency are computed using T_total. The spatial domain is $[0, p\pi] \times [0, \pi]$ with $h = \pi/256$, where p is the number of processors. The testing time interval is $[0, 1]$ with $\Delta t = 1/200$.

nprocs	T_total	T_comp	T_comm	Speedup	Efficiency	Max-Err
1	2.08e+01	2.08e+01	0.0e-02	1	100%	1.4e-03
2	2.12e+01	2.10e+01	1.1e-01	1.96	98.1%	1.4e-03
3	2.12e+01	2.10e+01	1.9e-01	2.94	98.1%	1.4e-03
4	2.12e+01	2.10e+01	1.5e-01	3.92	98.1%	1.4e-03
5	2.12e+01	2.10e+01	1.7e-01	4.91	98.1%	1.4e-03
6	2.13e+01	2.10e+01	2.5e-01	5.86	97.7%	1.4e-03
7	2.13e+01	2.11e+01	2.9e-01	6.84	97.7%	1.4e-03
8	2.13e+01	2.11e+01	2.2e-01	7.81	97.7%	1.4e-03
10	2.14e+01	2.11e+01	3.1e-01	9.72	97.2%	1.4e-03
12	2.14e+01	2.10e+01	4.1e-01	11.7	97.2%	1.4e-03
14	2.14e+01	2.10e+01	3.7e-01	13.6	97.2%	1.4e-03
16	2.14e+01	2.10e+01	3.8e-01	15.6	97.2%	1.4e-03
20	2.15e+01	2.10e+01	4.8e-01	19.3	96.7%	1.4e-03
24	2.15e+01	2.11e+01	4.3e-01	23.2	96.7%	1.4e-03
28	2.16e+01	2.11e+01	5.0e-01	27.0	96.3%	1.4e-03
32	2.16e+01	2.10e+01	5.6e-01	30.8	96.3%	1.4e-03
48	2.12e+01	2.09e+01	3.3e-01	47.1	98.1%	1.4e-03
64	2.15e+01	2.10e+01	5.1e-01	61.9	96.7%	1.4e-03
128	2.14e+01	2.10e+01	4.4e-01	124	97.2%	1.4e-03

Table 10

Solving $u_t = \Delta u + \sin(x)u_x - \cos(x)u$ by SEIDD3. Speedup and efficiency are computed using T_total.The spatial domain is $[0, p\pi] \times [0, \pi]$ with $h = \pi/256$, where p is the number of processors. The testing time interval is $[0, 1]$ with $\Delta t = 1/200$.

nprocs	T_total	T_comp	T_comm	Speedup	Efficiency	Max-Err
1	9.89e+00	9.89e+00	0.0e-02	1.00	100%	8.4e-04
2	1.04e+01	1.02e+01	1.9e-01	1.90	95.1%	8.4e-04
3	1.04e+01	1.02e+01	2.3e-01	2.85	95.1%	8.4e-04
4	1.05e+01	1.02e+01	2.9e-01	3.77	94.2%	8.4e-04
5	1.05e+01	1.02e+01	3.4e-01	4.71	94.2%	8.4e-04
6	1.06e+01	1.02e+01	3.5e-01	5.60	93.3%	8.4e-04
7	1.05e+01	1.02e+01	3.1e-01	6.59	94.2%	8.4e-04
8	1.05e+01	1.02e+01	2.8e-01	7.54	94.2%	8.4e-04
10	1.05e+01	1.02e+01	2.7e-01	9.92	94.2%	8.4e-04
12	1.05e+01	1.02e+01	3.0e-01	11.3	94.2%	8.4e-04
14	1.07e+01	1.01e+01	5.5e-01	12.9	92.4%	8.4e-04
16	1.06e+01	1.02e+01	4.1e-01	14.9	93.3%	8.4e-04
20	1.05e+01	1.01e+01	4.1e-01	18.8	94.2%	8.4e-04
24	1.05e+01	1.01e+01	4.0e-01	22.6	94.2%	8.4e-04
28	1.06e+01	1.02e+01	3.5e-01	26.1	93.3%	8.4e-04
32	1.07e+01	1.01e+01	5.5e-01	29.6	92.4%	8.4e-04
48	1.09e+01	1.00e+01	8.7e-01	43.6	90.7%	8.4e-04
64	1.07e+01	1.01e+01	5.6e-01	59.2	92.4%	8.4e-04
128	1.08e+01	1.01e+01	6.7e-01	117	91.6%	8.4e-04

SEIDD1 algorithm, while the communication time consumed by the two algorithms are the same. This difference in computation time is well under expectation since, as analyzed in section 6, the SEIDD3 has a linear order computation cost, while the SEIDD1 algorithm using an FFT-based subdomain solver has an $O(N \log N)$ computation cost on a grid with N grid points [42].

8. Concluding remarks. We developed a class of stabilized explicit-implicit domain decomposition methods by adding a stabilization step to the explicit-implicit domain decomposition methods. The EIDD methods are globally noniterative, nonoverlapping domain decomposition methods, which are computationally and communicationally efficient for each time step when compared with Schwarz-method-based parabolic solvers. However, EIDD methods suffer from either stability- or consistency-related time step size restrictions, while Schwarz methods could maintain the good stability condition of implicit temporal discretization schemes. The proposed SEIDD methods have inherited the advantages of EIDD methods in time-stepwise efficiency while exhibiting excellent stability experimentally. But more importantly for parallel computing, especially for large-scale simulation problems, the SEIDD methods achieve good stability without adding any communication cost or measurable computation cost to the EIDD methods. As confirmed by tests on SGI Origin 2000 computers, the excellent parallel speedup and scalability of the SEIDD methods suggest potential for large-scale simulation on massively parallel machines.

Acknowledgments. We would like to thank Professor X.-C. Cai, Professor David Keyes, and two anonymous reviewers for their comments and suggestions. We also would like to thank Professor Frank Neubrander for clarifying the exact meaning of an unstable dynamical system. The authors are grateful to the National Center for Supercomputing Applications (NCSA) for providing access to its SGI Origin 2000 parallel computer.

REFERENCES

[1] B. Bäumer and F. Neubrander, *Laplace transform methods for evolution equations*, Conf. Semin. Mat. Univ. Bari, 259 (1994), pp. 27–60.

[2] G. Birkhoff and R. E. Lynch, *Numerical Solution of Elliptic Problems*, SIAM, Philadelphia, 1984.

[3] K. Black, *Polynomial collocation using a domain decomposition solution to parabolic PDE's via the penalty method and explicit/implicit time marching*, J. Sci. Comput., 7 (1992), pp. 313–338.

[4] X.-C. Cai, *Additive Schwarz algorithms for parabolic convection-diffusion equations*, Numer. Math., 60 (1991), pp. 41–61.

[5] X.-C. Cai, *Multiplicative Schwarz methods for parabolic problems*, SIAM J. Sci. Comput., 15 (1994), pp. 587–603.

[6] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems*, Numer. Linear Algebra Appl., 1 (1994), pp. 477–504.

[7] X.-C. Cai and O. B. Widlund, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 243–258.

[8] X.-C. Cai and O. B. Widlund, *Multiplicative Schwarz algorithms for some nonsymmetric and indefinite problems*, SIAM J. Numer. Anal., 30 (1993), pp. 936–952.

[9] H. Chen and R. Lazarov, *Domain splitting algorithm for mixed finite element approximations to parabolic problems*, East-West J. Numer. Math., 4 (1996), pp. 121–135.

[10] J. Crank and P. Nicolson, *A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type*, Proc. Cambridge Philos. Soc., 43 (1947), pp. 50–67.

[11] D. S. Daoud, A. Q. M. Khaliq, and B. A. Wade, *A non-overlapping implicit predictor-corrector scheme for parabolic equations*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Vol. I, Las Vegas, NV, H.R. Arabnia et al., eds., CSREA Press, 2000, pp. 15–19.

[12] T. F. Chan and T. Mathew, *Domain decomposition algorithms*, Acta Numer., 1994, pp. 61–143.

[13] C. N. Dawson and T. F. Dupont, *Explicit/implicit, conservative domain decomposition procedures for parabolic problems based on block-centered finite difference*, SIAM J. Numer.

Anal., 31 (1994), pp. 1045–1061.

[14] C. Dawson, Q. Du, and T. Dupont, *A finite difference domain decomposition algorithm for numerical solution of the heat equation*, Math. Comp., 57 (1991), pp. 63–71.

[15] J. Douglas, *Alternating direction implicit methods for three space variables*, Numer. Math., 4 (1962), pp. 41–63.

[16] J. Douglas and J. Gunn, *A general formulation of alternating direction method: Part* I. *Parabolic and hyperbolic problems*, Numer. Math., 6 (1964), pp. 428–453.

[17] M. Dryja, *Substructuring methods for parabolic problems*, in Proceedings of the Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations (Moscow, 1990), SIAM, Philadelphia, 1991, pp. 264–271.

[18] M. Dryja and O. B. Widlund, *An Additive Variant of the Schwarz Alternating Method for the Case of Many Subregions*, Tech. Rep. 339, Courant Institute, New York University, 1987.

[19] E. Du Fort and S. Frankel, *Stability conditions in the numerical treatment of parabolic differential equations,* Math. Tables and Other Aids to Computation, 7 (1953), pp. 135–152.

[20] D. Funaro, *A multidomain spectral approximation of elliptic equations*, Numer. Methods Partial Differential Equations, 2 (1986), pp. 187–205

[21] W. D. Gropp and D. E. Keyes, *Domain decomposition on parallel computers*, Impact Comput. Sci. Engrg., 1 (1989), pp. 421–439.

[22] D. E. Keyes, *Domain Decomposition: A Bridge between Nature And Parallel Computers*, NASA ICASE Technical Report 92-44, NASA Langley Research Center, Hampton, VA, 1992.

[23] 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. Statist. Comput., 8 (1987), pp. 166–202.

[24] Y. A. Kuznetsov, *New algorithms for approximate realization of implicit difference schemes*, Soviet J. Numer. Anal. Math. Modelling, 3 (1988), pp. 99–114.

[25] Yu. A. Kuznetsov, *Domain decomposition methods for unsteady convection-diffusion problems*, in Proceedings of the 9th International Conference on Computing Methods in Applied Sciences and Engineering, Paris, 1990, R. Glowinski and A. Lichnewsky, eds., SIAM, Philadelphia, 1990, pp. 211–227.

[26] Yu. M. Laevsky, *A domain decomposition algorithm without overlapping subdomains for the solution of parabolic equations*, Comput. Math. Math. Phys., 32 (1992), pp. 1569–1580.

[27] Yu. M. Laevsky, *Explicit-implicit domain decomposition method for solving parabolic equations*, in Computing Methods and Technology for Solving Problems in Mathematical Physics, Ross. Akad. Nauk Sibirsk. Otdel., Vychisl. Tsentr, Novosibirsk, 1993, pp. 30–46 (in Russian).

[28] Y. M. Laevsky and S. V. Gololobov, *Explicit-implicit domain decomposition methods for the solution of parabolic equations*, Siberian Math. J., 36 (1995), pp. 506–516.

[29] Yu. M. Laevsky and O. V. Rudenko, *Splitting methods for parabolic problems in nonrectangular domains*, Appl. Math. Lett., 8 (1995), pp. 9–14.

[30] P. Lax and R. Richtmyer, *Survey of the stability of linear finite difference*, Comm. Pure Appl. Math., 9 (1956), pp. 267–293.

[31] T. P. Mathew, P. L. Polyakov, G. Russo, and J. Wang, *Domain decomposition operator splittings for the solution of parabolic equations*, SIAM J. Sci. Comput., 19 (1998), pp. 912–932.

[32] D. W. Peaceman and H. H. Rachford, Jr., *The numerical solution of parabolic and elliptic differential equations*, J. Soc. Ind. Appl. Math., 3 (1955), pp. 28–41.

[33] H. A. Schwarz, *Gesammelte Mathematische Abhandlungen*, Vol. 2, Springer-Verlag, Berlin, 1890, pp. 133–143.

[34] B. F. Smith, P. E. Bjorstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, Cambridge, UK, 1996.

[35] X.-H. Sun and L. Ni, *Scalable problems and memory-bounded speedup*, J. Parallel and Distributed Computing, 19 (1993), pp. 27–37.

[36] J. Zhu, *On an efficient parallel algorithm for solving time dependent partial differential equations*, in Proceeding of the 11th International Conference on Parallel and Distributed Processing Techniques and Applications, CSREA Press, Las Vegas, NV, 1998, pp. 394–401.

[37] Y. Zhuang, *Toward Understanding the Von Neumann Stability Condition*, preprint.

[38] Y. Zhuang, *Classically Unstable Approximations for Linear Evolution Equations and Applications*, mathematics Ph.D. dissertation, Louisiana State University, Baton Rouge, LA, 2000.

[39] Y. Zhuang, *A Class of Stable, Globally Non-iterative, Non-overlapping Domain Decomposi-*

tion Algorithms for the Parallel Simulation of Parabolic Evolutionary Systems, computer science Ph.D. dissertation, Louisiana State University, Baton Rouge, LA, 2000.

[40] Y. Zhuang and X.-H. Sun, *A domain decomposition based parallel solver for time dependent differential equations*, in Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX, 1999, CD-ROM SIAM, Philadelphia, 1999.

[41] Y. Zhuang and X.-H. Sun, *A high-order ADI solver for separable generalized Helmholtz equations*, Adv. Engrg. Software, 31 (2000), pp. 585–591.

[42] Y. Zhuang and X.-H. Sun, *A high order fast direct solver for singular poisson equations*, J. Comput. Phys., 171 (2001), pp. 79–94.

[43] Y. Zhuang and X.-H. Sun, *Stable, globally non-iterative, nonoverlapping domain decomposition parallel solvers for parabolic problems*, in Proc. of SuperComputing 2001 (SC2001), Denver, CO, 2001, CD-ROM, IEEE Computer Society Press, Los Alamitos, CA, Association for Computing Machinery, New York, 2001.