Parallel Performance Investigation of a Domain Decomposition Algorithm

W. Rivera*, J. Zhu†, and D.H. Huddleston‡

1 Introduction

Convection-diffusion equations in the form of

$$u_t + \alpha \nabla u = \nabla \cdot (\beta \nabla u), \quad x \in \Omega, \quad t > 0, \tag{1}$$

$$u(x,t) = f(x,t), \qquad x \in \partial\Omega, \qquad t > 0,$$

$$u(x,0) = g(x), \qquad x \in \Omega,$$

where Ω is the spatial domain and $\partial\Omega$ is the boundary of Ω, are widely used in science and engineering as mathematical models for computational simulations, such as in oil reservoir simulations, analysis of flow field around airplanes, transport of solutes in groundwater, and global weather prediction. In particular, when $\beta = 0$, equation (1) becomes a pure convection equation.

For large-scale problems, particularly those defined in two- or three-dimensional spatial domains, the computation of solutions may require substantial CPU time. It is therefore desirable to use multiprocessor parallel computers to calculate solutions. A widely used method for solving time dependent PDEs on parallel computers is domain decomposition [2]. It dates back to the classical Schwarz alternating algorithm with overlapping subdomains [12] for solving elliptic boundary value problems. Note

*Wilson Rivera, Electrical and Computer Engineering Department, University of Puerto Rico at Mayaguez (wrivera@ece.uprm.edu).

†Jianping Zhu: Department of Mathematics and Statistics, Mississippi State University, (jzhu@erc.msstate.edu).

‡David H. Huddleston, Department of Civil Engineering, Mississippi State University, (hudd@erc.msstate.edu).

that the original motivation for using domain decomposition method was to deal with complex geometries, equations that exhibit different behaviors in different regions of the domain, and memory restriction for solving large scale problems.

When solving time dependent PDEs with non-overlapping subdomains on parallel computers, the domain decomposition method could either be used as a preconditioner for Krylov type algorithms [2], or as a means to decompose the original domain into subdomains and solve the PDEs defined in different subdomains concurrently [4]. When it is used as a preconditioner, the relevant PDE is discretized over the entire original domain to form a large system of algebraic equations, which is then solved by Krylov type iterative algorithms. The preconditioning step and the inner products involved in the solution process often incur a significant amount of communication overhead that could significantly affect the scalability of the solution algorithms.

On the other hand, if the original domain Ω is decomposed into a set of non-overlapping subdomains $\Omega_k, k = 1, \cdots, M$, it would be ideal that the PDEs defined in different subdomains could be solved on different processors concurrently. This often requires numerical boundary conditions at the boundaries between subdomains. These numerical boundary conditions are not part of the original mathematical model and the physical problem. One way to generate those numerical boundary conditions is to use the solution values from the previous time step t_n to calculate the solutions at t_{n+1} [1, 8]. This is often referred to as time lagging (TL). The other way to generate numerical boundary conditions is to use an explicit algorithm to calculate the solutions at the boundaries between subdomains, using the solutions from the previous time step, and then solve the PDEs defined on different subdomains concurrently using an implicit method [3, 6]. This is referred to as the explicit predictor (EP) method in this paper. In an earlier paper [9], the authors showed, in the context of the numerical solution of one-dimensional linear eat equation, that the stability and accuracy of the solution algorithm can be significantly affected by the TL and EP methods. A new approach based on explicit predictors and implicit correctors (EPIC) for the solution of convection-diffusion equations was proposed in another previous paper [10]. The results demonstrated a significant improvement in accuracy when calculating transient solutions. In this paper a systematic investigation of parallel performance and scalability for this new approach will be presented. The next section is devoted to the description of the different methods for generating numerical boundary conditions between subdomains. The parallel implementation and performance models will be discussed in Section 3. Parallel performance for the solution of Euler equations will be presented in Section 4, followed by the conclusions in Section 5.

2 Domain Decomposition and Numerical Boundary Conditions

For simplicity of the discussion, the following one-dimensional linear model with constant coefficients and homogeneous boundary conditions is used here to describe

different methods for generating numerical boundary conditions:

$$\begin{aligned} &u_t + \alpha u_x = \beta u_{xx}, \quad 0 < x < 1, \quad 0 < t \leq T, \\ &u(0,t) = u(1,t) = 0, \quad t > 0, \\ &u(x,0) = g(x), \quad 0 \leq x \leq 1. \end{aligned} \tag{2}$$

The original spatial domain $\Omega = [0,1]$ is discretized by a set of grid points $x_i, i = 0, \ldots, L$, uniformly distributed with $\Delta x = x_i - x_{i-1} = \frac{1}{L}$. The temporal domain $[0,T]$ is discretized by a set of discrete time steps $t_n, n = 0, \cdots, N$, with $\Delta t = t_n - t_{n-1} = \frac{T}{N}$. The numerical solution $u(x_i, t_n)$ is denoted by u_i^n, and the original spatial domain is decomposed into M subdomains $\Omega_k, k = 1, \ldots, M$, where the two end points of subdomain Ω_k are denoted as r_{k-1} and r_k, respectively. Each subdomain Ω_k has $m+1$ points including the two end points r_{k-1} and r_k. Since only two physical boundary conditions are available at the points r_0 and r_M, numerical boundary conditions are needed at points r_k, $k = 1, \cdots, M-1$, if the PDEs defined in different subdomains are to be solved concurrently using an implicit algorithm.

Various finite difference algorithms are available for discretizing equation (2). For example, the forward time central difference (FTCS) scheme given is by

$$\begin{aligned} &u_i^{n+1} = (r + \frac{R}{2})u_{i-1}^n + (1-2r)u_i^n + (r - \frac{R}{2})u_{i+1}^n, \\ &i = 1, ..., L-1, \quad n = 0, ..., N-1, \end{aligned} \tag{3}$$

where $R = \alpha \frac{\Delta t}{\Delta x}$ and $r = \beta \frac{\Delta t}{\Delta x^2}$, and the implicit backward time central difference (BTCS) scheme is given by

$$\begin{aligned} &-(r + \frac{R}{2})u_{i-1}^{n+1} + (1+2r)u_i^{n+1} - (r - \frac{R}{2})u_{i+1}^{n+1} = u_i^n, \\ &i = 1, ..., L-1, \quad n = 0, ..., N-1. \end{aligned} \tag{4}$$

2.1 Time-Lagging (TL) Method

For the TL method, the boundary conditions between subdomains are generated by setting

$$\begin{aligned} \bar{u}_{r_k-1}^{n+1} &= u_{r_k-1}^n, \\ \bar{u}_{r_k}^{n+1} &= u_{r_k}^n, \qquad k = 1, \ldots, M-1. \end{aligned} \tag{5}$$

Note that the left side of subdomain $\Omega_k, k = 2, \ldots, M$, is extended to $r_{k-1} - 1$ in order to advance the solution value at the point r_{k-1} to the next time level. An implicit scheme is then used to solve the PDE in each subdomain concurrently.

The method is stable and causes an additional truncaton error of order $\mathcal{O}[\frac{(\Delta t)}{(\Delta x)}]$.

2.2 Explicit-Predictor (EP) Method

For the EP method with central difference for both the convection and diffusion term, the algorithm can be written as

- Predictor

$$\bar{u}_{r_k}^{n+1} = (r + \frac{R}{2})u_{r_k-1}^n + (1-2r)u_{r_k}^n + (r - \frac{R}{2})u_{r_k+1}^n, \quad k = 1, \cdots, M-1, \tag{6}$$

- Calculation of solutions in each subdomain

$$\begin{bmatrix} a_0 & a_1 & 0 & \cdots & 0 \\ a_2 & a_0 & a_1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_1 \\ 0 & \cdots & 0 & a_2 & a_0 \end{bmatrix} \begin{bmatrix} u_{r_{k-1}+1}^{n+1} \\ u_{r_{k-1}+2}^{n+1} \\ \vdots \\ u_{r_k-2}^{n+1} \\ u_{r_k-1}^{n+1} \end{bmatrix} = \begin{bmatrix} u_{r_{k-1}+1}^n - a_2\bar{u}_{r_{k-1}}^{n+1} \\ u_{r_{k-1}+2}^n \\ \vdots \\ u_{r_k-2}^n \\ u_{r_k-1}^n - a_1\bar{u}_{r_k}^{n+1} \end{bmatrix}, \tag{7}$$

$$k = 1, \cdots, M,$$

where $a_0 = 1 + 2r$, $a_1 = -(r - \frac{R}{2})$, and $a_2 = -(r + \frac{R}{2})$.

The method is only conditionally stable. For example, when

$R \geq 1$, the scheme is unstable. The additional error caused by the EP method with central difference is dominated by the term of order $\mathcal{O}[\frac{\Delta t^2}{\Delta x}]$.

2.3 Explicit-Predictor Implicit-Corrector (EPIC)

The EPIC method combines the advantages of both the TL (stability) and EP (accuracy) methods. The following are the main steps of the EPIC method with central difference for both the convection and diffusion terms:

- Use an explicit predictor, such as the FTCS algorithm, to generate the numerical boundary conditions at the end points r_k, $k = 1, \cdots, M-1$, of all subdomains:

$$\bar{u}_{r_k}^{n+1} = (r + R)u_{r_k-1}^n + (1-2r)u_{r_k}^n + (r - R)u_{r_k+1}^n, \quad k = 1, ..., M-1, \tag{8}$$

- Solve the systems of equations in all subdomains Ω_k, $k = 1, \cdots, M-1$, concurrently:

$$\begin{bmatrix} a_0 & a_1 & 0 & \cdots & 0 \\ a_2 & a_0 & a_1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_1 \\ 0 & \cdots & 0 & a_2 & a_0 \end{bmatrix} \begin{bmatrix} u_{r_{k-1}+1}^{n+1} \\ u_{r_{k-1}+2}^{n+1} \\ \vdots \\ u_{r_k-2}^{n+1} \\ u_{r_k-1}^{n+1} \end{bmatrix} = \begin{bmatrix} u_{r_{k-1}+1}^n - a_2\bar{u}_{r_{k-1}}^{n+1} \\ u_{r_{k-1}+2}^n \\ \vdots \\ u_{r_k-2}^n \\ u_{r_k-1}^n - a_1\bar{u}_{r_k}^{n+1} \end{bmatrix}, \tag{9}$$

- Update the numerical boundary conditions at points r_k, $k = 1, \cdots, M-1$, using an implicit corrector, such as the BTCS algorithm:

$$u_{r_k}^{n+1} = \frac{(r+R)}{(1+2r)}u_{r_k-1}^n + \frac{1}{(1+2r)}u_{r_k}^n + \frac{(r-R)}{(1+2r)}u_{r_k+1}^{n+1}. \tag{10}$$

The method is stable and causes an additional truncation error of order $\mathcal{O}[\frac{\Delta t^2}{\Delta x}]$.

3 Parallel Implementation

In the following discussion we assume ideal conditions for load balancing, that is, the number of subdomains is equal to the number of processors, and the number of grid points is roughly equal in all subdomains.

In the first step of the computation, each processor calculates the numerical boundary condition(s) needed for its subdomain using an explicit method, such as (4). Note that the first and the last processor only need to calculate one numerical boundary condition, while other processors need to calculate two numerical boundary conditions. These computations can be done concurrently on all processors.

In the second step, each processor forms the system of linear algebraic equations similar to that in (4) using the numerical boundary conditions calculated at the first step, and then solves the system of equations. These computations can also be done concurrently on all processors.

After the solutions have been calculated, each processor must send to and receive from its neighboring processors the solutions at the points next to the end points of the subdomain. For the processor holding subdomain Ω_k with the end points r_{k-1} and r_k, it must send the calculated solutions at $r_{k-1}+1$ and r_k-1 to the left and right neighboring processors, respectively, and receive the newly calculated solutions at the points $r_{k-1}-1$ and r_k+1 from the left and right neighboring processors, respectively. Note that the first and the last processors on the chain only need to communicate with one neighboring processor. This is the only communication step involved in the EPIC method.

In the last step of the computation, each processor corrects the numerical boundary condition(s) at the end of its subdomain using an implicit method, such as (4). This can again be done in parallel.

The message passing standard MPI [5] is used in the code implementation to ensure maximum portability to a wide range of architectures, including both distributed and shared memory parallel computers, as well as clusters of workstations and personal computers.

4 Parallel Performance

In order to simplify the discussion in the previous sections, we have considered a one-dimensional convection-diffusion model. However, to evaluate parallel performance of the methods we will consider the solution of the Euler equations in the flow calculation around a NACA0012 airfoil.

The algorithm applied to the entire domain without domain decomposition is an implicit finite volume formulation, first order accurate in time with an approximate Riemann solver based on Roe flux approximation to achieve up to third order spatial accuracy as reported by Whitfield et al. [13],

For parallel processing, The MacCormack scheme [7] is used as the predictor method; and the Euler solver based on the Roe's scheme [13] is used to calculate solutions in all subdomains concurrently, and to update the boundary data between subdomains.

The algorithms have been tested on an SGI Power Challenge XL parallel

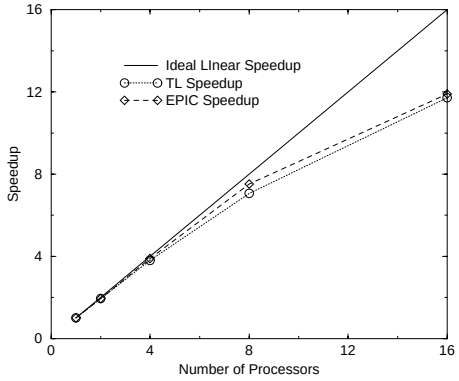

Figure 1. *Speedup:* 129×31 *grid for the NACA0012 airfoil*

computer with 16 R8000 processors. The following set of figures compare the performance of the TL algorithm vs. the EPIC algorithm. We do not consider the EP algorithm in the discussion because there is no significant difference between the EP and EPIC algorithms due to the corrector step.

Figure 1 shows the speedup obtained using a coarse 129×31 C-grid for the computation of the steady state solution at Mach number $M_\infty = 0.85$ and angle of attack $\alpha = 1.0$. Speedup is defined as the ratio between the sequential execution time and the parallel execution time. For fixed size problems, speedup is limited because of the overhead which grows with increasing number of processors or because the number of processors exceeds the degree of concurrence of the algorithm, that is, the maximum number of tasks which can be executed simultaneously.

As a consequence the problem size should be increased in order to achieve an improvement in performance. Efficiency, which is the ratio between the sequential execution time and the cost of the parallel system, may be maintained constant by increasing the problem size as the number of processors increases.

Figure 2 shows the speedup curves for a finer 290×81 C-grid. A better performance of the EPIC algorithm can be noted.

It is clear from the figures that the EPIC algorithm scales well as the number of processors increases for large scale problems. Even a slight superlinear speedup can be observed from the Figure 2. This is mainly due to the nonuniform access latency for different levels of cache and memory on SGI Power Challenge. When more processors are used, the array sizes of the code on each processor become smaller, which preserves data locality better and results in more efficient utilization of local cache and memory.

The steady-state solution of a problem requires a sufficient number of time steps to reduce the solution residual to a small value. The number of time steps to obtain the steady state solution depends on the solution technique, the size of the problem, the number of subdomains, and the desired level of convergence of the solution. This suggests a tradeoff between accuracy of the solution and performance. The problem size should be increased as the number of subdomains

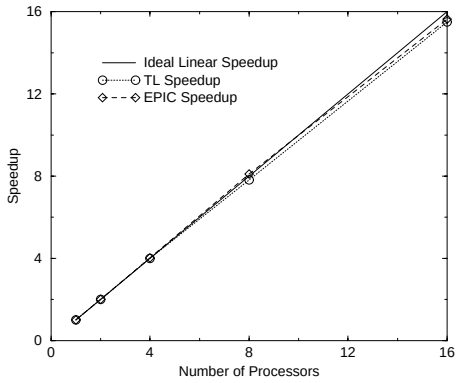

Figure 2. *Speedup:* 290×81 *grid for the NACA0012 airfoil*

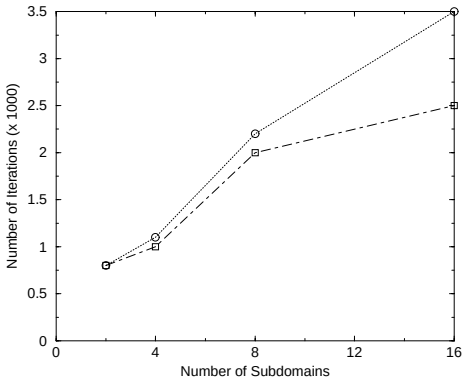

Figure 3. *Influence of number of subdomains on the solution convergence:* $\circ - -\circ$ *TL Method and* $\diamond - \cdot - \diamond$ *EPIC Method.*

increases to achieve an improvement on performance, and on the other hand, the increase in the number of subdomains and problem size have an important impact on the solution convergence properties.

Figure 3 shows the number of iterations needed to reduce the residual to order 10^{-8} with regard to the number of subdomains. More iterations are necessary for the TL method compared to the EPIC method to reduce the solution residual as the number of subdomains increases.

5 Conclusions

A systematic investigation of parallel performance and scalability for a domain decomposition algorithm based on explicit predictors and implicit correctors has been presented. This method has demonstrated a significant improvement in accuracy when calculating transient solutions of time dependent partial differential

equations. The results in this paper show that the new approach scales well as the number of processors increases for large scale problems. In addition, we have shown the tradeoff between accuracy of the solution and performance of the parallel implementation.

Bibliography

[1] S. Barnard, S. Saini, R. Van der Wijngaart, M. Yarrow, L. Zechtzer, I. Foster, and O. Larsson, "Large-scale distributed computational fluid dynamics on the information power grid using Globus," in Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, 1999.

[2] T. F. Chan and T. P. Mathew, "Domain decomposition algorithms," *Acta Numerica,* 3 (1994), pp. 61-143.

[3] C. N. Dawson, Q. Du, and T. F. Dupont, "A finite difference domain decomposition algorithm for numerical solution of the heat equations," *Mathematics of Computation*, **57** (1991), pp. 63 - 71.

[4] D. Drikakis and E. Schreck, "Development of parallel implicit Navier-Stokes solvers on MIMD multi-processor systems," *AIAA* 93-0062.

[5] W. Gropp, M. Snir, B. Nitzberg, E. Lusk, "MPI: The Complete Reference," MIT Press, Cambridge, 1998.

[6] Y. A. Kuznetsov, "New algorithms for approximate realization of implicit difference schemes," *Sovietic Journal of Numerical Analysis and Mathematical Modeling,* 3 (1988), pp. 99-114.

[7] R. W. MacCormack, "The effects of viscosity in hypervelocity impact cratering," AIAA 69-354, 1969.

[8] R. Pankajakshan and W. R. Briley, "Parallel solution of viscous incompressible flow on multi-block structured grids using MPI," *Parallel Computational Fluid Dynamics: Implementation and Results Using Parallel Computers,* A. Ecer, J. Periaux, N. Satofuka, and S. Taylor, ed., Elsevier Science, Amsterdam, 1996, pp. 601-608.

[9] W. Rivera and J. Zhu, "A scalable parallel domain decomposition algorithm for solving time dependent partial differential equations," *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Technology and Applications*, H. R. Arabnia Ed., CSREA Press, Athens, GA 1999, pp. 240-246.

[10] W. Rivera, J. Zhu and D. Huddleston, "An efficient parallel algorithm with application to computational fluid dynamics," *To appear in Computers & Mathematics with Applications.*

[11] P. L. Roe, "Approximate Riemann solvers, parameter vector, and difference schemes," *Journal of Computational Physics*, vol. 43, pp. 357–372, 1981.

[12] H.W. Schwarz, "Gesammelete Mathematische Abhandlungen," 2 (1890), pp. 133-143.

[13] D. L. Whitfield, J. M. Janus, and L. B. Simpson, "Implicit finite volume high resolution wave split scheme for solving the unsteady three-dimensional Euler and Navier-Stokes equations on stationary or dynamic grids," Engineering and Industrial Research Report MSSU-EIRS-ASE-88-2, Mississippi State University, 1988.