LA-UR 97-1605

CONF-970876--1

*Approved for public release; distribution is unlimited*

Title: The Delayed Coupling Method: An Algorithm for Solving Banded Diagonal Matrix Problems in Parallel



Author(s): N. Mattor
T. J. Williams
D. W. Hewett
A. M. Dimits

Submitted to: IMACS '97
August 24-29, 1997
Berlin, Germany



**Los Alamos**
**National Laboratory**

## DISCLAIMER

# The Delayed Coupling Method: An Algorithm for Solving Banded Diagonal Matrix Problems in Parallel

N. Mattor, T. J. Williams[1], D. W. Hewett, A. M. Dimits
Lawrence Livermore National Laboratory
Livermore, California 94550 USA
e–mail: mattor@m5.llnl.gov

## ABSTRACT

We present a new algorithm for solving banded diagonal matrix problems efficiently on distributed–memory parallel computers, designed originally for use in dynamic alternating–direction implicit (ADI) partial differential equation solvers. The algorithm optimizes efficiency with respect to the number of numerical operations, and with respect to the amount of interprocessor communication. We refer to our approach as the "delayed coupling method" because the communication is deferred until needed. We focus here on tridiagonal and periodic tridiagonal systems.

## 1. INTRODUCTION

We discuss a new approach to parallel solution of banded linear systems, the "delayed coupling method." The method is analogous to the solution of an inhomogeneous linear differential equation, where the solution is a "particular" solution added to an arbitrary linear combination of "homogeneous" solutions. The coefficients of the homogeneous solutions are later determined by boundary conditions. In our parallel method, each processor is given a contiguous subsection of a tridiagonal system. With no information about the neighboring subsystems, each processor obtains the solution up to two constants. Then the global solution can be found by matching endpoints.

Our earlier paper[1] has a more detailed description of the method and its application to tridiagonal systems. The algorithm is designed with the following objectives, listed in order of priority. The first objective is to minimize the number of interprocessor communications opened, since this is the most time consuming process. Second, the algorithm allows flexibility of the specific solution method of the tridiagonal submatrices. Here, we employ a variant of LU decomposition, but this is easily replaced with cyclic reduction or other. Third, we wish to minimize storage needs.

## 2. BASIC ALGORITHM

We consider the $N \times N$ tridiagonal linear system

$$\Lambda \mathbf{X} = \mathbf{R}, \tag{1}$$

[1]Now at Los Alamos National Laboratory

with

$$\Lambda = \begin{pmatrix} B_1 & C_1 & & & & \\ A_2 & B_2 & C_2 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & C_{N-1} \\ & & & & A_N & B_N \end{pmatrix},$$

$$\begin{aligned} \mathbf{X} &= (X_1 \quad X_2 \quad \ldots \quad X_N)^T, \\ \mathbf{R} &= (R_1 \quad R_2 \quad \ldots \quad R_N)^T, \end{aligned}$$

on a parallel computer with $P$ processors. For simplicity, we assume $N = PM$, with $M$ an integer.

Our algorithm is as follows. First, we divide the linear system of order $N$ into $P$ subsystems of order $M$. Thus, the $N \times N$ matrix $\Lambda$ is divided into $P$ submatrices $L_p$, each of dimension $M \times M$,

$$\Lambda = \begin{pmatrix} L_1 & c_M^1 \mathbf{e}_M \mathbf{e}_1^T & & & \\ a_1^2 \mathbf{e}_1 \mathbf{e}_M^T & L_2 & c_M^2 \mathbf{e}_M \mathbf{e}_1^T & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & c_M^{P-1} \mathbf{e}_M \mathbf{e}_1^T \\ & & & a_1^P \mathbf{e}_1 \mathbf{e}_M^T & L_P \end{pmatrix},$$

where $\mathbf{e}_p$ is the $p^{th}$ column of the $M \times M$ identity matrix. Similarly, we divide the $N$ dimensional vectors $\mathbf{X}$ and $\mathbf{R}$ into $P$ sub–vectors $\mathbf{x}$ and $\mathbf{r}$, each of dimension $M$

$$\begin{aligned} \mathbf{X} &= \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_P \end{pmatrix}^T, \\ \mathbf{R} &= \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \ldots & \mathbf{r}_P \end{pmatrix}^T. \end{aligned}$$

For each subsystem $p$ we define three vectors $\mathbf{x}_p^R$, $\mathbf{x}_p^{UH}$, and $\mathbf{x}_p^{LH}$ as the solutions to the equations

$$L_p \mathbf{x}_p^R \equiv \mathbf{r}_p, \tag{2}$$

$$L_p \mathbf{x}_p^{UH} \equiv (-a_1^p \quad 0 \quad 0 \quad \ldots \quad 0)^T, \tag{3}$$

$$L_p \mathbf{x}_p^{LH} \equiv (0 \quad 0 \quad 0 \quad \ldots \quad -c_M^p)^T. \tag{4}$$

The superscripts on the $\mathbf{x}$ stand for "particular," "upper homogeneous," and "lower homogeneous" solution respectively, from the inhomogeneous differential equation analogy. Here $a_m^p$ is the $m^{th}$ subdiagonal element of the $p^{th}$ submatrix, *etc.*

The general solution of subsystem $p$ is $\mathbf{x}_p^R$ plus an arbitrary linear combination of $\mathbf{x}_p^{UH}$ and $\mathbf{x}_p^{LH}$,

$$\mathbf{x}_p = \mathbf{x}_p^R + \xi_p^{UH} \mathbf{x}_p^{UH} + \xi_p^{LH} \mathbf{x}_p^{LH}, \tag{5}$$

where $\xi_p^{UH}$ and $\xi_p^{LH}$ are yet undetermined coefficients that depend on coupling to the neighboring solutions. To find $\xi_p^{UH}$ and $\xi^{LH}$, substitute Eq. (5) into Eq. (1). Straightforward calculation shows

that $\xi_1^{UH} = \xi_P^{LH} = 0$, and the remaining $2P-2$ coefficients are determined by the solution to the following tridiagonal linear system, or "reduced" system:

$$\begin{pmatrix} x_{1,M}^{LH} & -1 & & & & & & & \\ -1 & x_{2,1}^{UH} & x_{2,1}^{LH} & & & & & & \\ & x_{2,M}^{UH} & x_{2,M}^{LH} & -1 & & & & & \\ & & -1 & x_{3,1}^{UH} & x_{3,1}^{LH} & & & & \\ & & & x_{3,M}^{UH} & x_{3,M}^{LH} & -1 & & & \\ & & & & \cdot & \cdot & \cdot & & \\ & & & & & \cdot & \cdot & \cdot & \\ & & & & & & \cdot & \cdot & -1 \\ & & & & & & & -1 & x_{P,1}^{UH} \end{pmatrix} \begin{pmatrix} \xi_1^{LH} \\ \xi_2^{UH} \\ \xi_2^{LH} \\ \xi_3^{UH} \\ \xi_3^{LH} \\ \cdot \\ \cdot \\ \xi_{P-1}^{LH} \\ \xi_P^{UH} \end{pmatrix} = - \begin{pmatrix} x_{1,M}^{R} \\ x_{2,1}^{R} \\ x_{2,M}^{R} \\ x_{3,1}^{R} \\ x_{3,M}^{R} \\ \cdot \\ \cdot \\ x_{P-1,M}^{R} \\ x_{P,M}^{R} \end{pmatrix}, \quad (6)$$

where $x_{p,m}$ refers to the $m^{th}$ element of the appropriate solution from the $p^{th}$ submatrix.

## 3. COMPUTING THE PARTICULAR AND HOMOGENEOUS SOLUTIONS

We find the three solutions $\mathbf{x}_p^R$, $\mathbf{x}_p^{UH}$, and $\mathbf{x}_p^{LH}$ by solving Eqs. (2)-(4). Exploiting overlapping calculations and elements with value 0 gives the following algorithm, with $13M$ binary floating point operations:

*Forward elimination:*

$$\begin{aligned} \omega_1 &= \frac{c_1}{b_1} & \omega_i &= \frac{c_i}{b_i - a_i\omega_{i-1}} & i &= 2,3,\ldots M \\ \gamma_1 &= \frac{r_1}{b_1} & \gamma_i &= \frac{r_i - a_i\gamma_{i-1}}{b_i - a_i\omega_{i-1}} & i &= 2,3,\ldots M \end{aligned}$$

*Back substitution:*

$$\begin{aligned} x_M^R &= \gamma_M & x_i^R &= \gamma_i - \omega_i x_{i+1}^R & i &= M-1, M-2, \ldots 1 \\ x_M^{LH} &= -\omega_M & x_i^{LH} &= -\omega_i x_{i+1}^{LH} & i &= M-1, M-2, \ldots 1 \\ \omega_M^{UH} &= \frac{a_M}{b_M} & \omega_i^{UH} &= \frac{a_i}{b_i - c_i\omega_{i+1}^{UH}} & i &= M-1, M-2, \ldots 1 \end{aligned}$$

*Forward substitution:*

$$x_1^{UH} = -\omega_1^{UH} \qquad x_i^{UH} = -\omega_i^{UH} x_{i-1}^{UH} \qquad i = 2,3,\ldots M,$$

where the processor index $p$ is implicitly present on all variables, and end elements $a_1$ and $c_M$ are written in the appropriate positions in the $a$ and $c$ arrays. The sample code in ref. [1] implements this with no temporary storage arrays.

## 4. CONSTRUCTION AND SOLUTION OF THE REDUCED MATRIX

Once each processor has determined $\mathbf{x}_p^R$, $\mathbf{x}_p^{UH}$, and $\mathbf{x}_p^{LH}$, we construct and solve the reduced system of Eq. (6). We assume that the following functions are available for interprocessor communication:

- `Send(ToPid,data,n)`: When invoked by processor `FromPid`, the array `data` of length `n` is sent to processor `ToPid`. `Send()` is *nonblocking*.
- `Receive(FromPid,data,n)`: To complete data transmission, processor `ToPid` invokes `Receive()`. Upon execution, the array sent by processor `FromPid` is stored in the array `data` array of length `n`. `Receive()` is *blocking* (the processor waits for the data to be received before continuing).

Opening interprocessor communications is generally the most time–consuming step in the entire tridiagonal solution process, so it is important to minimize this. The following algorithm consumes a time of $T = (\log_2 P)t_c$ in opening communication channels (where $t_c$ is the time to open one channel).

1. Each processor writes whatever data it has that is relevant to Eq. (6) in the array `OutData`.
2. The `OutData` arrays from each processor are concatenated as follows (Fig. 1):
   (a) Each processor $p$ sends its `OutData` array to processor $p - 1 \pmod P$, and receives a corresponding array from processor $p + 1 \pmod P$, as depicted in Fig. 1a. The incoming array is concatenated to the end of `OutData`.
   (b) At the $i^{th}$ step, repeat the first step, except sending to processor $p-2^{i-1} \pmod P$, and receiving from processor $p+2^{i-1} \pmod P$ (Fig. 1b,c), for $i = 1, 2, \ldots$. After $\log_2 P$ iterations (or the next higher integer), each processor has the contents of the reduced matrix in the `OutData` array.
3. Each processor rearranges the contents of its `OutData` array into a local copy of the reduced tridiagonal system, and then solves. At this point, each processor has all the values in Eq. (5) stored locally.

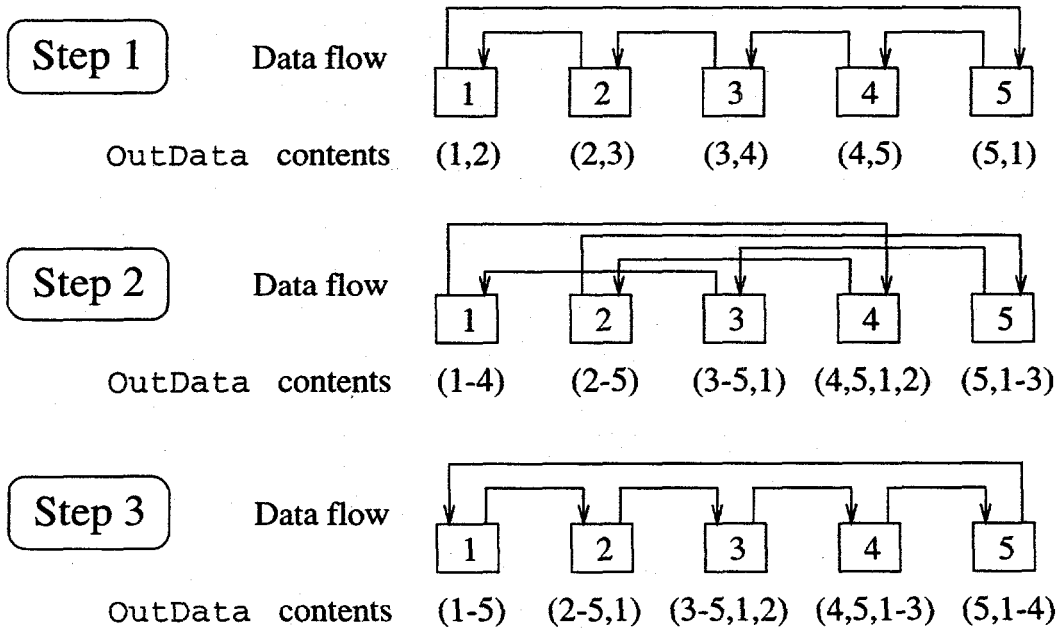


Figure 1: *Illustration of the method to pass reduced matrix data between processors, shown for* $P = 5$.

## 5. PERFORMANCE

The time consumption for this routine is as follows:

1. To calculate the three roots $\mathbf{x}^R$, $\mathbf{x}^{UH}$, and $\mathbf{x}^{LH}$ requires $13M$ binary floating point operations by each processor, done in parallel.
2. To assemble the reduced matrix in each processor requires $\log_2 P$ steps where interprocessor communications are opened, and the $i^{th}$ opening passes $8 \times 2^{i-1}$ real numbers.
3. Solution of the reduced system through LU decomposition requires $8\,(2P-2)$ binary floating point operations by each processor, done in parallel.
4. Calculation of the final solution requires $4M$ binary floating point operations by each processor, done in parallel.

If $t_b$ is the time of one binary floating point operation, $t_c$ is the time required to open a communication channel (latency), and $t_p$ is the time to pass one real number once communication is opened, then the time to execute this parallel routine is given by (optimally)

$$\begin{aligned} T_P &\simeq 13Mt_b + (\log_2 P)\, t_c + 8\,(P-1)\, t_p + 8\,(2P-2)\, t_b + 4Mt_b \\ &\simeq (17M + 16P)\, t_b + (\log_2 P)\, t_c + 8Pt_p, \end{aligned} \tag{7}$$

for $P \gg 1$. For cases of present interest, $T_P$ is dominated by $(\log_2 P)t_c$ and $17Mt_b$. The *parallel efficiency* is defined by $\epsilon_P \equiv \frac{T_S}{PT_P}$, where $T_S$ is the execution time of a serial code which solves by LU decomposition. Serial LU decomposition solves an $N \times N$ system in a time $T_S = 8Nt_b$, so

$$\epsilon_P = \frac{8}{17 + 16P^2/N + (\log_2 P)\, Pt_c/Nt_b + 8P^2 t_p/Nt_b}. \tag{8}$$

To test these claims empirically, we measured the execution times of working serial and parallel codes, and calculated $\epsilon_P$ both through its definition and through Eq. (8). Fig. 2 shows $\epsilon_P$ as a function of $P$ for two cases, $N = 200$ and $N = 50,000$. We conclude from Fig. 2 that Eq. (8) (smooth lines) is reasonably accurate, both for the theoretical maximum efficiency (47%, achieved for small $P$ and large $N$) and for the scaling with large $P$.

We made these timings on the BBN TC2000 machine at Lawrence Livermore National Laboratory, using 64-bit floating point arithmetic. This machine had 128 M88100 RISC processors, connected by a butterfly–switch network. To calculate the predictions of Eq. (7) we chose $t_c = 750\mu$sec, based on the average time of a send/receive pair measured in our code; based on other measurements, we chose the passage time of a single 64-bit real number as $t_p = 9\mu$sec; we chose $t_b = 1.4\mu$sec, based on our measured timing of 0.00218 sec for the serial algorithm on the $N = 200$ case.

## 6. PERIODIC TRIDIAGONAL SYSTEM

We have generalized our algorithm to a "periodic tridiagonal system." This is a tridiagonal system with additional nonzero elements in the far upper and lower corners of the matrix, that is, Eq. (1)

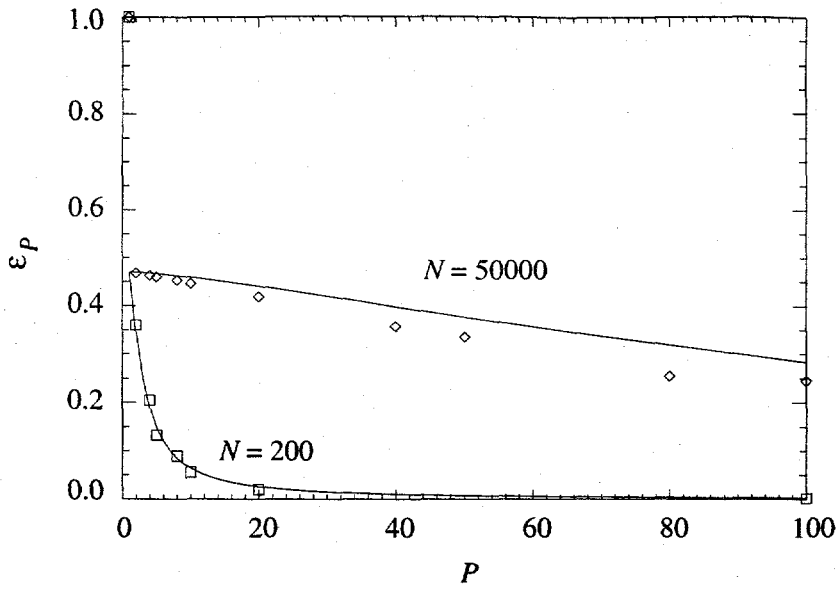


Figure 2: *Results of scaling runs, comparing the parallel time with serial LU decomposition time. Here, $\epsilon_P$ is the parallel efficiency and $P$ is the number of processors. The smooth lines represent Eq. (8), and the points are empirical results.*

with $\Lambda$ now of the form

$$\Lambda = \begin{pmatrix} B_1 & C_1 & & & & A_1 \\ A_2 & B_2 & C_2 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & C_{N-1} \\ C_N & & & & A_N & B_N \end{pmatrix}. \tag{9}$$

Solution proceeds almost precisely as before. First divide Equation (9) into tridiagonal subsystems, and solve for the particular, upper, and lower homogeneous solutions. The subsystems are again all tridiagonal, so no additional consideration need be given to this part. Then use these solutions to construct a reduced system analogous to Eq. (6). Here, however, the first and last subsystems acquire drives for the upper and lower homogeneous solutions, respectively, so the condition $\xi_1^{UH} = \xi_P^{UH} = 0$ no longer pertains. This leads to a reduced matrix with nonzero elements in the far corners

$$\begin{pmatrix} x_{1,M}^{LH} & -1 & & & & & & & \\ -1 & x_{2,1}^{UH} & x_{2,1}^{LH} & & & & & x_{1,M}^{UH} & \\ & x_{2,M}^{UH} & x_{2,M}^{LH} & -1 & & & & & \\ & & -1 & x_{3,1}^{UH} & x_{3,1}^{LH} & & & & \\ & & & x_{3,M}^{UH} & x_{3,M}^{LH} & -1 & & & \\ & & & & \cdot & \cdot & \cdot & & \\ & & & & & \cdot & \cdot & \cdot & \\ & & & & & & \cdot & \cdot & -1 \\ x_{P,1}^{LH} & & & & & & -1 & x_{P,1}^{UH} & \end{pmatrix} \begin{pmatrix} \xi_1^{LH} \\ \xi_2^{UH} \\ \xi_2^{LH} \\ \xi_3^{UH} \\ \xi_3^{LH} \\ \cdot \\ \cdot \\ \xi_{P-1}^{LH} \\ \xi_P^{UH} \end{pmatrix} = - \begin{pmatrix} x_{1,M}^{R} \\ x_{2,1}^{R} \\ x_{2,M}^{R} \\ x_{3,1}^{R} \\ x_{3,M}^{R} \\ \cdot \\ \cdot \\ x_{P-1,M}^{R} \\ x_{P,M}^{R} \end{pmatrix}.$$

This system necessitates a new solution algorithm. The most efficient we know (not shown here) uses $LU$ decomposition, and requires $15P$ binary operations. The interesting consequence is that the parallel efficiency nearly doubles over the nonperiodic case, since the operation count in the corresponding serial solver also rises—from $8N$ to $15N$. Thus, Eq. (8) for the predicted efficiency becomes

$$\epsilon_P = \frac{15}{17 + 16P^2/N + (\log_2 P)\, Pt_c/Nt_b + 15P^2 t_p/Nt_b}.$$

## 7. DISCUSSION AND CONCLUSIONS

Stability of the parallel tridiagonal algorithm is similar to that of serial LU decomposition of a tridiagonal matrix. If the $L_i$ are unstable to LU decomposition, then pivoting could be used. If the $L_i$ are singular, then LU decomposition fails and some alternative should be devised. If the large matrix $\Lambda$ is diagonally dominant, then so too are the $L_i$. If the reduced system is unstable to LU decomposition, this can be replaced by a different solution scheme, with little loss of overall speed (if $P \ll M$).

This routine is generalizable from tridiagonal to higher systems. For example, in a 5-diagonal system, there would be four homogeneous solutions, each with an undetermined coefficient. The coefficients of the homogeneous solutions would be determined by a reduced system analogous to Eq. (6), except with $O(4P)$ equations, not $2P - 2$.

In our applications of the parallel tridiagonal solver we solve a tridiagonal system along each line of grid points parallel to a given direction. In two or higher dimensions, each processor owns a segment of each of many systems, giving us a strong advantage in interprocessor communication over solving only a single system: each processor solves *all* of its triplets of independent subsystems, then packs together all of the data it needs to send to other processors—there is only one send volley for solving all of the systems. Furthermore, that number of processors each processor communicates with is the number of processors collinear in the one direction, which will generally be smaller than the total number of processors in a multidimensional domain decomposition; this improves parallel efficiency by reducing the value of $P$ below the total number of processors.

This work was performed for the U.S. Department of Energy at Lawrence Livermore National Laboratory under contract W–7405–ENG–48 and Los Alamos National Laboratory under contract W–7405–ENG–36.

## REFERENCES

[1] N. Mattor, T. J. Williams, D. W. Hewett, *Algorithm for solving tridiagonal matrix problems in parallel*, Parallel Computing 25, p. 1769 (1995).

[2] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, Adam Hilger, Bristol, p. 185 (1988).