# Parallel Implementation of 2-Dimensional Toeplitz Solver on MasPar with Applications to Image Restoration

Kin-wai Mak[*] Raymond H. Chan[†]

**Abstract**

Image restoration problems can be transformed into problems of solving a linear system $Tx = b$ where $T$ is a block-Toeplitz or near-block-Toeplitz matrix. However, for many of these problems, the size of the matrix $T$ is usually very large. For instance, if we are going to manipulate a 1024-by-1024 pixel image, then $T$ will be of the order $1024^2$-by-$1024^2$. In this paper, we implement a parallel version of our existing 2-Dimensional Toeplitz solver in a data-parallel fashion on MasPar (DECmpp) by fully utilizing its massively parallel processing power. The package we developed is portable and easy-to-use. We will demonstrate how to use our package to recover a satellite image which is blurred by atmospheric turbulence. The implementation details and performance results will also be presented.

**Key Words.** High Performance Fortran, Data Parallel, Toeplitz matrix, Circulant matrix.

## 1 Introduction

In image restoration, the problems can be transformed into problems of solving a linear system $Tx = b$ where $T$ is a block-Toeplitz or near-block-Toeplitz matrix. However, for many of these problems, the size of the matrix $T$ is usually very large. For instance, if we are going to manipulate a 1024-by-1024 pixel image, then $T$ will be of the order $1024^2$-by-$1024^2$. For such systems, our existing sequential 2-Dimensional Toeplitz solver written in MATLAB is already very fast but it still cannot deblur large images in real time. Our chief goal in this paper is to implement a *Parallel 2-D Toeplitz solver* that is portable and easy to use.

We will first recall an implementation of our *Parallel 1-D Toeplitz solver* on MasPar — a massively parallel computer. Our results show that the $O(n \log n)$ sequential algorithm can be reduced to an $O(\log n)$ algorithm provided that sufficient number of processors is used. This leads us to the implementation of the parallel version of our *2-D Toeplitz solver* on MasPar in this paper.

To guarantee portability, we choose *High Performance Fortran* (HPF) as the main programming language. The parallelization is done in a data-parallel fashion. In our design, we fully exploit the use of *shell script*, *fmex file* and *PVM*. The main idea is to build a bridge using the *fmex file* so that M-files (MATLAB programs) resided in a workstation can communicate with the HPF executable resided in MasPar through PVM. The shell script encapsulated in the M-file of our package is just used for starting up the daemon of PVM automatically. We remark that we just use PVM for sending

[*]Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong.

[†]Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong. Research supported in part by HK Research Grant Council grants no. CUHK316/94E.

data between a workstation and MasPar. All the main computation and parallelization are done by the HPF executable resided in MasPar. Using this approach, we can have the visualization power of MATLAB together with the parallelization power of HPF.

The outline of the paper is as follows. In §2, we give the mathematical background of Toeplitz solvers. In §3, we give the parallel implementation of our 2-D Toeplitz solver in details. Numerical results of an application to ground-base astronomy are given in §4 to illustrate that our design is indeed efficient. Finally, concluding remarks and acknowledgment are given in §5 and §6 respectively.

## 2 Toeplitz Solvers

In this section, we give some mathematical background for implementing our 1-D and 2-D Toeplitz solvers. Let us review the definitions of Toeplitz and circulant matrices first. An $n$-by-$n$ matrix $T_n$ is said to be *Toeplitz* if it is of the form:

$$T_n = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{2-n} & t_{1-n} \\ t_1 & t_0 & t_{-1} & & t_{2-n} \\ \vdots & t_1 & t_0 & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{bmatrix}.$$

An $n$-by-$n$ matrix $C_n$ is said to be *circulant* if it has the form:

$$C_n = \begin{bmatrix} c_0 & c_{-1} & c_{-2} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{-1} & \cdots & c_3 & c_2 \\ c_2 & c_1 & \ddots & \ddots & & c_3 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{-2} & c_{-3} & & \ddots & \ddots & c_{-1} \\ c_{-1} & c_{-2} & c_{-3} & \cdots & c_1 & c_0 \end{bmatrix}.$$

One of the nice features of circulant matrices is that they can be diagonalized by Fourier matrix $F_n$. In other words, $C_n = F_n^* \Lambda_n F_n$ where $\Lambda_n$ is a diagonal matrix holding the eigenvalues of $C_n$. Thus, for any vector $x$, the matrix-vector multiplication $C_n x$ can be done in $O(n \log n)$ operations by using Fast Fourier Transform (FFT). In particular, we can compute the matrix-vector multiplication $T_n x$ in $O(n \log n)$ operations as well by simply embedding $T_n$ into a $2n$-by-$2n$ circulant matrix and then using FFT, see for instance [3].

For the 1-dimensional case, to solve the linear system $T_n x = b$ where $T_n$ is an $n$-by-$n$ Toeplitz matrix, we use the preconditioned conjugate gradient (PCG) method with T. Chan's circulant preconditioner. The construction of T. Chan's circulant preconditioner $C_n$ of $T_n$ is defined as follows:

$$c_j = \begin{cases} \dfrac{(n-j)t_j + j t_{j-n}}{n}, & 0 \le j < n, \\ c_{n+j}, & 0 < -j < n \end{cases}$$

where $c_j$ and $t_j$ are the $j$-th diagonals of $C_n$ and $T_n$ respectively, see [3].

In Table 1, the PCG method is outlined. The steps labeled with ($\star$) are highly parallelizable because the matrix-vector multiplication can be handled by using 1-D FFT. Other steps like vector updating and inner products can also be easily parallelized using data-parallel paradigm.

```
$x = 0$
$r = b$
$\rho = 1$
$p = 0$
For $i = 1, 2, \ldots$
    $z = C^{-1} r$            $(\star)$
    $\rho' = \rho$
    $\beta = \rho / \rho'$
    $p = z + \beta p$
    $q = T p$                 $(\star)$
    $\alpha = \rho / p^t q$
    $x = x + \alpha p$
    $r = r - \alpha p$
End
```

**Table 1 :** PCG method

As an example for evaluating the performance of our *Parallel 1-D Toeplitz solver*, we choose $T_n$ to be the $n$-by-$n$ Toeplitz matrix generated by the generating function $\theta^4 + 1$ and solve the linear system $T_n x = b$ where $b$ is an $n$-by-1 vector with all entries equal to $1/\sqrt{n}$, see for instance [4]. From Figure 1, we found that the $O(n \log n)$ sequential algorithm can basically be reduced to an $O(\log n)$ algorithm provided that the number of processors used is at least $2n$. This suggests us to implement our 2-D Toeplitz solver.

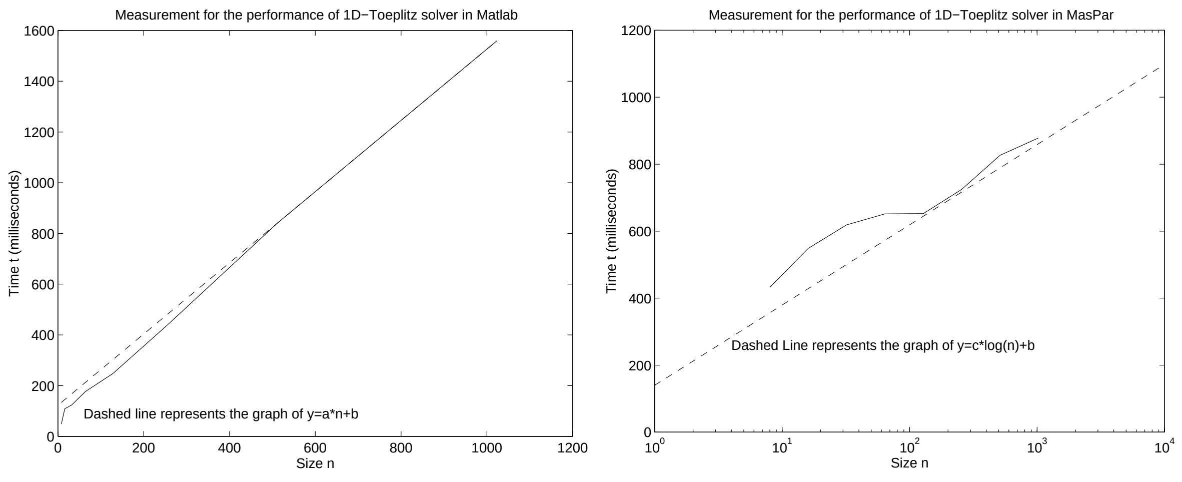

**Figure 1.** Sequential Algorithm (Left) and Parallel Algorithm (Right)

For the 2-dimensional case, we consider solving the linear system $T_n x = b$ where $T_n$ is now an $n^2$-by-$n^2$ block-Toeplitz-Toeplitz-block (BTTB) matrix, see [2]. We can still use the PCG method to solve such linear system with the matrix-vector multiplications handled by 2-D FFT. The construction of T. Chan's circulant preconditioner of $T_n$ is slightly different from that in the 1-dimensional case, see for instance [2]. We leave the parallel implementation details in the next section.

## 3 Parallel Implementation of 2-D Toeplitz Solver

### 3.1 Overview of MasPar

In this section, we give some basic features about the parallel machine we are using — the **DECmpp 12000/Sx**, where "mpp" stands for massively parallel processors or simply MasPar. MasPar is basically a single instruction multiple data (SIMD) machine, i.e. every processor executes the same instruction but on different data points. The model we are using is an MP-1 which consists of totally 8192 processors arranged in a mesh topology. Its architecture can be divided into two parts :

1. **Front End**
   This is a processor running ULTRIX as the operating system. It is mainly for executing instructions on singular data. In particular, it supports the communication with the data parallel unit.

2. **Data Parallel Unit**
   Data Parallel Unit (**DPU**) is a unit responsible for all of the parallel processing. It consists of an array control unit (**ACU**), a processor element (**PE**) array as well as a global router. The main function of **ACU** is for controlling the **PE** array and sending data or instructions to each **PE** simultaneously. For the **PE** array, it consists of 8192 processors arranged in a toroidal mesh. Each processor has 64KB local memory available for use. As for the global router, it governs one way of the communications between **PEs** in the **PE** array.

### 3.2 Design Methodology

There are many different ways to do parallelization. For instance, one may consider connecting a cluster of workstations using message passing libraries like MPI or PVM. However, it may not be desirable to do so as the implementation may be rather complicated and the performance may not be good enough since communication between the workstations can be very slow. In our design, we use *High Performance Fortran* (HPF) as the main programming language for the implementation, and the main computation is done in MasPar which is massively parallel. As HPF is emerging to be a prominent and robust language, we believe that the portability of our implementation can be enhanced.

Unfortunately, there is no easy way to do visualization using HPF alone. One can simply let the HPF program to write the resulting data in a data file and then get it back to MATLAB for visualization. However, we do not want to do this separately. Instead, we want to have an integrated environment. In other words, we would like to enjoy the visualization power of MATLAB and the parallelization power of HPF simultaneously.

In MATLAB, there is a nice tool called *fmex file* which provides a channel for M-files (MATLAB programs) to call fortran (F77) programs directly. Thus, our idea is to build a bridge or a wrapper using fmex file so as to fill in the gap between the M-files resided in a workstation and an HPF program resided in MasPar, i.e., we have to make the fmex file to interface with the HPF program.

This can be done because there is a construct called *interface* and *end interface* in HPF for interfacing with F77 subroutines. The problem remains to solve is how to pass data from the bridge (fmex file) to the HPF executable resided in MasPar or vice versa. This can be accomplished by using PVM. We emphasize that we just make use of PVM for data input and data output only. Using the directive *cmpf f77* in HPF, the HPF program can recognize those PVM subroutines as F77 programs. For instance, if we add a statement *cmpf f77 PVMFSEND* in an HPF code, then the HPF program will recognize the *PVMFSEND*, a built-in PVM subroutine, as a F77 program during compilation. In such a way, the bridge can be built. We have not mentioned the details about fmex and PVM. The readers can refer to [5, 11].

We remark that in our implementation, the fmex file and PVM subroutines are almost transparent to the programmers and the users. In fact, Siu and Tam [7] have built a utility for easy generation of fmex files, and have collected some of the built-in PVM subroutines as a library. Using their library, we can simply add a few more lines in our HPF program for interfacing with the fmex file and PVM subroutines. More precisely, to build a parallel solver for solving $T_N x = b$ where $T_N$ is an $N$-by-$N$ BTTB matrix and $N = n^2$, the following lines are just what we need to add in our HPF program:

- `include 'hpf.int'`
- `call getrealmatrix(b,N,1)`
- `call getrealmatrix(T,N,1)`
- `call putrealmatrix(x,N,1)`

Here, `hpf.int` is the header file containing all the declarations of the PVM subroutines to be used. The functions `getrealmatrix(b,N,1)`, `getrealmatrix(T,N,1)` and `putrealmatrix(x,N,1)` are HPF subroutines to be interfaced with PVM. The former two will get an $n^2$-by-1 real vector `b` and an $n^2$-by-1 real vector `T` (the first column of $T_N$) from MATLAB. The latter one will transfer an $n^2$-by-1 real vector `x` (the solution) to MATLAB for visualization.

To sum up, our approach is a hybrid one. The package we developed contains four main components, namely an M-file, an fmex file, the PVM environment and an HPF program. The control flow is as follows (see Figure 2):

1. The M-file (run.m) will call a fmex file (solver.mexrs6).
2. The fmex file will call PVM subroutines for sending messages to MasPar.
3. PVM will invoke a slave program (solver) which is an HPF executable resided in MasPar.
4. PVM will pack and send the input data to the HPF program (solver) for performing the main computation.
5. Results obtained from the HPF executable (solver) will be packed and passed back to the fmex file via PVM.
6. The fmex file will pass back the received results to the M-file (run.m) for visualization.

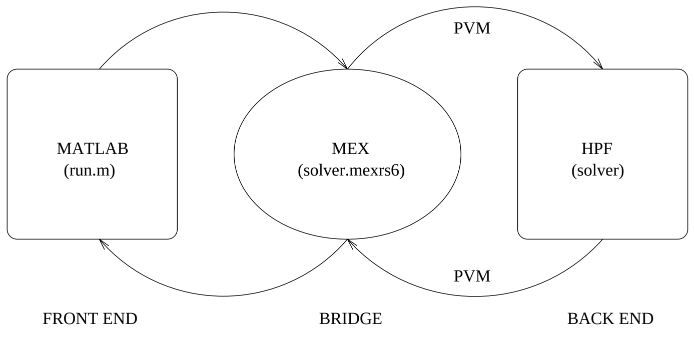

**Figure 2.** Flow chart of the package

## 3.3 Implementation Details

In this section, we introduce the techniques of using HPF for doing parallelization in a data-parallel fashion. Let us give an overview of HPF first. *High Performance Fortran* (HPF) is simply an extension of Fortran 90. It is mainly for implementing single program multiple data (SPMD) programming model. Using parallel directives, the programmer can do parallelization for their applications more easily. It is becoming a standard programming language for different kinds of parallel computers.

The HPF we are using in MasPar is called *mpfortran* which is a subset of HPF. In order to parallelize the PCG method using mpfortran, all we have to do is to spread all the data over the DPU of MasPar. Once the data are on DPU, the execution can be done in parallel over different data points. In our implementation, we distribute the data in a (cyclic,cyclic) manner, i.e., if we distribute an $n$-by-$n$ matrix $A=(a_{ij})$ over the PE array in DPU, then the entries $a_{ij}$ will be assigned to $P_{(i \bmod 128, j \bmod 64)}$ where $P_{(i,j)}$ denotes the $(i,j)$-th processor in the PE array. There is no need to worry about if the size of the matrix $A$ is greater than the size of the PE array because the virtualization is done by the compiler automatically. Here, we list several different ways for forcing the data to be on DPU:

- `cmpf ONDPU A` $\Rightarrow$ Force A to be on DPU using ONDPU directive
- `forall (i=1:10,j=1:10) A(i,j)=1.d0` $\Rightarrow$ Use forall construct
- `A(1:10,1:10) = 1.d0` $\Rightarrow$ Use Fortran 90 style (i.e. vectorization)
- `where (A` $\geq$ `0) A = 1.d0` $\Rightarrow$ Use mask assignment

We remark that translating mpfortran codes to FULL HPF codes is very easy. In general, we need to add back some data mapping directives that are available in FULL HPF but are absent in mpfortran. For more details on HPF, see [6, 8, 9].

The remaining step in our implementation is to find a way to call the parallel version of FFT subroutines available in MasPar. In MasPar, those parallel FFT subroutines are available from the MasPar Mathematics Library (MPML). A compilation flag *-lmpml* is needed during compilation. The FFT routines that we need to use are complex-to-complex 2-D FFT and the calling syntax is as follows:

```
call cfft2d_sizes(N,N,wss1,wps1)
call cfft2d_init(ws1,wp1,N,N)
call cfft2d(A,ws1,wp1,N,N,buffer)
```

The result of the above-mentioned calls will be stored in the matrix `A` finally. The 2-D inverse FFT calls are of the same calling style as above, see [10] for details.

## 4 Application to Ground-Base Astronomy

As an application of our 2-D Toeplitz solver, we consider a problem in ground-base astronomy. We are given an observed satellite image and we want to recover the original satellite image. The blurring function is approximated by using a guide star image, see [1]. This leads to a linear system of equations $Hx = b$ where $H$ is the blurring function which is estimated from the guide star image and $b$ is a vector obtained from the observed satellite image, see Figure 3. However, $H$ is a rectangular block-circulant-circulant-block (BCCB) matrix. Hence, we have to solve its normal equation instead, i.e., $H^*Hx = H^*b$. Since $H^*H$ is very ill-conditioned, we employ the Tikhonov regularization which results in solving:

$$(\lambda I + H^*H)x = H^*b$$

where $\lambda$ is the regularization parameter. Here, $(\lambda I + H^*H)$ is an $n^2$-by-$n^2$ block-Toeplitz-Toeplitz-block matrix and therefore we use our *Parallel 2-D Toeplitz solver* to solve it.

We remark that the only data inputs needed here are the $n$-by-$n$ matrix `H` (the guide star image) and the $n$-by-$n$ matrix `b` (the observed image) which are used for the construction of the first column of the matrix $H$ and the vector $b$ respectively. Plug in the design methodology of our parallel solver mentioned in §3.2, we make some suitable changes in the HPF program accordingly. The corresponding changes are as follows:

- `call getrealmatrix(b,N,1)` $\Rightarrow$ `call getrealmatrix(b,n,n)`
- `call getrealmatrix(T,N,1)` $\Rightarrow$ `call getrealmatrix(H,n,n)`
- `call putrealmatrix(x,N,1)` $\Rightarrow$ `call putintmatrix(x,n,n)`

The HPF executable *solver* and the fmex file *solver.mexrs6* are modified accordingly.

In our experiment, we choose an IBM 43P workstation as the front end to run our package and take a 256-by-256 pixel image for testing. Thus, we are solving a linear system of dimension $256^2$-by-$256^2$. However, by using our package, the recovered satellite image (see Figure 4) can be visualized within 26.6410 seconds. This is a speed up of 13.5130 times compared with that using the sequential solver written in MATLAB (see Table 2).

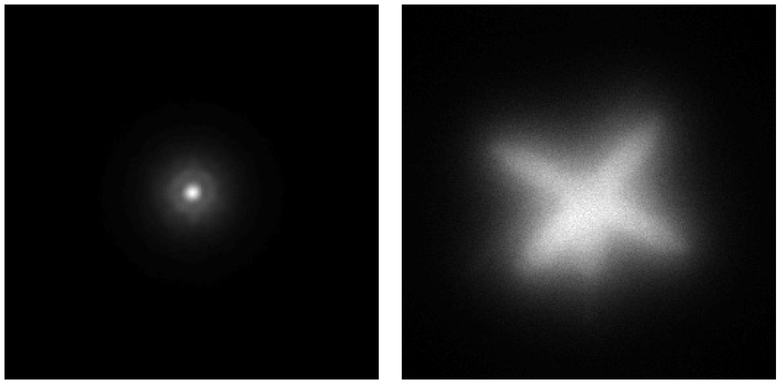

**Figure 3.** Guide Star Image (Left) and Observed Image (Right)

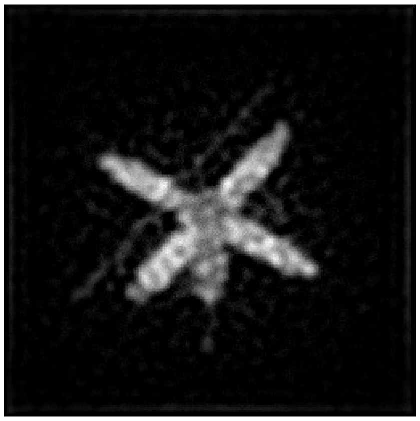

**Figure 4.** Recovered Image using Parallel Algorithm
(10 iterations with regularization parameter $\lambda = 10^{-4}$)

| Sequential Algorithm (Matlab) | Parallel Algorithm (HPF) | Speed Up |
|---|---|---|
| 360 sec | 26.6410 sec | 13.5130 |

**Table 2 :** Performance Measurement

## 5 Concluding Remarks

In this paper, we have presented a user-friendly and efficient parallel package for solving 2-D Toeplitz problems. From our numerical results, our *Parallel 2-D Toeplitz Solver* implemented on MasPar can almost deblur a satellite image in real time and a reasonable speed up is achieved. Our future work is to implement such parallel solver on other parallel computers like the Intel Paragon and the IBM SP2 which are multiple data multiple instructions (MIMD) machines. We are particularly interested in implementing our design on Shared Memory Parallel (SMP) machines as well. Performance analysis for our implementation on different kind of parallel machines will be carried out in our future work.

## 6 Acknowledgments

We would like to express our sincere thanks to Prof. Omar Wing, Dr. Danny Luk, Mr. Tommy Siu and Mr. Gary Tam for lending us the nice interface *MATLAB Interface for Massively Parallel Processing.*

## References

[1] R. Chan, M. Ng and R. Plemmons, *Generalization of Strang's Preconditioner with Applications to Toeplitz Least Squares Problems*, Numer. Linear Algebra Appls., 3 (1996), 45–64.

[2] R. Chan and X. Jin, *A Family of Block Preconditioners for Block Systems*, SIAM J. Sci. Stat. Comput., 13 (1992), 1218–1235.

[3] R. Chan and M. Ng, *Congugate Gradient Method for Toeplitz Systems*, SIAM Review, 38 (1996), 427–482.

[4] R. Chan, *Toeplitz Preconditioners for Toeplitz Systems with Nonnegative Generating Functions*, IMA J. of Numer. Anal., 11 (1991), 333–345.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.

[6] C. Koelbel, D. Loveman, R. Schriber, G. Steele Jr and M. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, 1994.

[7] T. Siu and G. Tam, *Matlab Interface for Massively Parallel Processsing*, private communication.

[8] *DECmpp Sx High Performance Fortran User Manual*, Digital Equipment Corporation, Mass., 1993.

[9] *DECmpp Sx High Performance Fortran User Guide*, Digital Equipment Corporation, Mass., 1993.

[10] *MasPar Mathematics Library (MPML) Reference Mannual*, Digital Equipment Corporation, Mass., 1993.

[11] *MATLAB External Interface Guide*, The Math Works Inc., Mass., 1993.