Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp290-295)

MPI Management of Hermite Collocation Computation on a Distributed-Shared Memory System

E. N. MATHIOUDAKIS* and E. P. PAPADOPOULOU†
Applied Mathematics and Computers Laboratory
Department of Sciences
Technical University of Crete
University Campus, 73100 Chania
GREECE

Abstract : - The organization and management of the intense computation involved in the solution of large, sparse and non-symmetric systems of equations, arising from the discretization of Elliptic Boundary Value Problems (BVPs) by the Hermite Collocation method, on a Distributed-Shared Memory (DSM) multiprocessor environment is the problem considered herein. As the size of the problem directly suggests the usage of iterative methods, we consider the parallel implementation of the Successive Over-Relaxation (SOR) and the preconditioned Bi-Conjugate Gradient Stabilized (Bi-CGSTAB) methods on DSM systems. Using the parallel algorithms we devised in [9, 10], which take advantage of the red-black structure of the Collocation matrix, we address the problem of efficiently managing the whole computation, through the MPI programming model, on a DSM system. The application was developed on a SGI Origin 350 DSM multiprocessor computer and its performance is revealed by the speedup measurements included.

Key-Words : - Collocation, SOR, preconditioned Bi-CGSTAB, Red-Black Orderings, MPI, DSM Systems.

1 Introduction

Hermite Collocation is a high order finite element scheme used as a discretizer especially when continuous first derivatives are required. Among other properties, Collocation produces large and sparse systems of equations which poses no pleasant properties (e.g. symmetry). Memory requirements and performance are two of the main factors suggesting the usage of iterative methods on multiprocessor environments. This motivated relevant research in the areas of iterative method analysis and parallel algorithm development. Main issues addressed were concerning both algorithmic (multi-color orderings, domain decomposition/partitioning techniques, parallel preconditioners, etc) and architectural (memory management/distribution, processor architecture, etc) aspects. Particular results, in this direction, concerning the Collocation method may be found in (e.g. [1,3,5,6,14] and in our work in [13,7,9,10]. It is worthwhile to mention that in [8,11], we conducted a performance analysis on a large family of Krylov subspace methods, including GMRES[15] and Bi-CGSTAB[17] as well as several preconditioning schemes, and concluded that from the tested Krylov methods the Backward Gauss-Seidel (B-GS) preconditioned Bi-CGSTAB-P yields fast rates of convergence when it applies to the solution of the Hermite Collocation Poisson system, while at the same time, outperforms all stationary iterative schemes, including SOR, for medium and large size problems. In [9] and [10] :

- We presented a generalized technique for devising efficient parallel algorithms, by constructing a virtual parallel machine ideally fitted for the problem at hand, and, in the sequel, by making use of partitioning techniques, the virtual computation was mapped onto a fixed size parallel architecture
- We effectively used this technique to devise efficient parallel algorithms for the iterative solution of Hermite Collocation systems, by the preconditioned Bi-CGSTAB and SOR methods.

In the work herein, to exploit and enhance the algorithm's parallelism, we implement the above methods on environments supporting *Incremental Parallelization* [2,4]. Namely, we implement the already appropriately designed parallel algorithms on DSM machines using the MPI standard [12], which utilize data dependency and memory allocation issues, as the Automatic Parallelization Option (APO) of the MipsPro compiler is unable to do so. Therefore we improve the algorithm's performance, on DSM machines, by managing the whole computation in order to maximize locality and minimize communication among the processing elements.

This paper is organized as follows: In Section 2 we briefly describe the structure of the collocation system and the iterative methods used, as well as we highlight the computationally intense parts where incremen-

Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp290-295)

tal parallelism is to be applied. We also incorporate the particular structure of the matrices involved into the algorithms. In Section 3, we present the basic features of the parallel architecture used to carry out the whole computation, together with the programming protocol each processor uses to compute intense matrix-vector operations according to the MPI standard. Finally, in Section 4, we present speedup measurements from the implementation on a SGI Origin 350 DSM system.

2 SOR / Bi-CGSTAB for Collocation

Let us consider the linear system

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

where the matrix A is in the well known red-black partitioning form

$$A = \begin{bmatrix} D_R & H_B \\ H_R & D_B \end{bmatrix}. \tag{2}$$

Collocation, among other celebrated discretizers such as the Finite Differences, through a particular numbering of equations and unknowns, results to a red-black ordered system when applied on Elliptic BVPs (e.g. [11]). And as it is shown in [10,11], both B-GS preconditioned Bi-CGSTAB and SOR iterative methods may be effectively used for its solution.

The Algorithms

By considering now the classical splitting

$$A = D_A - L_A - U_A \tag{3}$$

where $D_A = \begin{bmatrix} D_R & O \\ O & D_B \end{bmatrix}$, $L_A = \begin{bmatrix} O & O \\ -H_R & O \end{bmatrix}$ (4)

and

$$U_A = \begin{bmatrix} O & -H_B \\ O & O \end{bmatrix}, \tag{5}$$

the iterative methods under consideration may be algorithmically described by :

SOR

$\mathcal{M}_\omega = D_A - \omega L_A$
$\mathcal{E}_\omega = (1-\omega)D_A + \omega U_A$
Choose $\boldsymbol{x}^{(0)}$
for $i = 1, 2, \ldots$
  $\boldsymbol{t} = \mathcal{E}_\omega \boldsymbol{x}^{(i)}$
  $\boldsymbol{t} = \boldsymbol{t} + \omega\boldsymbol{b}$
  Solve $\mathcal{M}_\omega \boldsymbol{x}^{(i+1)} = \boldsymbol{t}$
  Check for Convergence
end

B-GS Preconditioned Bi-CGSTAB [17]

Choose $\boldsymbol{x}^{(0)}$
$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}$
Choose $\hat{\boldsymbol{r}}$ (usually $\hat{\boldsymbol{r}} = \boldsymbol{r}^{(0)}$)
for $i = 1, 2, \ldots$
  $\rho_{i-1} = \hat{\boldsymbol{r}}^T \boldsymbol{r}^{(i-1)}$
  if $\rho_{i-1} = 0$ method **fails**
  if $i = 1$
    $\boldsymbol{p}^{(1)} = \boldsymbol{r}^{(0)}$
  else
    $\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}} \frac{\alpha_{i-1}}{\omega_{i-1}}$
    $\boldsymbol{p}^{(i)} = \boldsymbol{r}^{(i-1)} + \beta_{i-1}(\boldsymbol{p}^{(i-1)} - \omega_{i-1}\boldsymbol{v}^{(i-1)})$
  endif
  Solve $\mathcal{M}\hat{\boldsymbol{p}} = \boldsymbol{p}^{(i)}$; $\boldsymbol{v}^{(i)} = A\hat{\boldsymbol{p}}$
  $\alpha_i = \frac{\rho_{i-1}}{\hat{\boldsymbol{r}}^T \boldsymbol{v}^{(i)}}$
  $\boldsymbol{s} = \boldsymbol{r}^{(i-1)} - \alpha_i \boldsymbol{v}^{(i)}$
  if $\|\boldsymbol{s}\|$ is small enough **then**
    $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \alpha_i\hat{\boldsymbol{p}}$ and **stop**
  Solve $\mathcal{M}\boldsymbol{z} = \boldsymbol{s}$; $\boldsymbol{t} = A\boldsymbol{z}$
  $\omega_i = \frac{\boldsymbol{s}^T\boldsymbol{t}}{\boldsymbol{t}^T\boldsymbol{t}}$
  $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \alpha_i\hat{\boldsymbol{p}} + \omega_i\boldsymbol{z}$
  Check for Convergence
  if $\omega_i = 0$ **stop**
  $\boldsymbol{r}^{(i)} = \boldsymbol{s} - \omega_i\boldsymbol{t}$
end

Notice that:
• The highlighted statements are the computationally intense parts of the algorithms
• The above algorithm for the Bi-CGSTAB implements the Bi-CGSTAB-P version[16], which minimizes the residual, instead of the preconditioned residual, and is equivalent to the postconditioned Bi-CGSTAB. For this case, in [11], we observed that the B-GS postconditioned Bi-CGSTAB yields faster convergence, hence

$$\mathcal{M} = D_A - U_A\,. \tag{6}$$

Incorporating the Collocation Structure

Aiming to the overall (serial and parallel) improvement of the computational performance, it is necessary to incorporate the particular structures, of the matrices involved, into the algorithms. For instance, in doing so for the intense

Solve $\mathcal{M}\boldsymbol{z} = \boldsymbol{s}$; $\boldsymbol{t} = A\boldsymbol{z}$

one may easily observe its equivalence, and therefore replace it into the algorithm, with the red-black instruction block:

  Solve $D_B\,\boldsymbol{z}_B = \boldsymbol{s}_B$
  $\boldsymbol{y} = H_B\,\boldsymbol{z}_B$

Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp290-295)

$$\begin{aligned}&\text{Solve } D_R\, \boldsymbol{z}_R = \boldsymbol{s}_R - \boldsymbol{y}\\ &\hat{\boldsymbol{y}} = H_R\, \boldsymbol{z}_R\\ &\boldsymbol{t}_R = \boldsymbol{s}_R\\ &\boldsymbol{t}_B = \boldsymbol{s}_B + \hat{\boldsymbol{y}}\end{aligned}$$

Furthermore, upon application of the Hermite Collocation method on Helmholtz-type Dirichlet BVPs, on a uniformly partitioned (into $n_s = 2p$ subintervals in both directions) unit square, the associated with relation (2) matrices $D_R, D_B, H_R, H_B \in \mathbb{R}^{8p^2,8p^2}$ are defined [10] by :

$$\begin{aligned}
D_R &= \text{diag}[\underbrace{A_2\ \ 2A_1\ \ 2A_2\ \cdots\ 2A_1\ \ 2A_2\ \ -A_2}_{2p-blocks}]\ ,\\
D_B &= 2\ \text{diag}[\underbrace{A_1\ \ A_2\ \cdots\ A_1\ \ A_2}_{2p-blocks}]\ ,\\
H_R &= [H_1^{(R)}\ \cdots\ H_{2p}^{(R)}]\ ,\\
H_B &= [H_1^{(B)}\ \cdots\ H_{2p}^{(B)}]\ ,
\end{aligned} \tag{7}$$

where each $H_j^{(R)} \in \mathbb{R}^{8p^2,4p}$ is defined by [16]:

$$\begin{aligned}
H_1^{(R)} &= (\boldsymbol{e}_1 - \boldsymbol{e}_2) \otimes A_4\\
H_{2p}^{(R)} &= -(\boldsymbol{e}_{2p-1} + \boldsymbol{e}_{2p}) \otimes A_4\ ,
\end{aligned}$$

and for $k = 1, \ldots, p-1$

$$\begin{aligned}
H_{2k}^{(R)} &= (\boldsymbol{e}_{2k-1} + \boldsymbol{e}_{2k} + \boldsymbol{e}_{2k+1} - \boldsymbol{e}_{2k+2}) \otimes A_3\\
H_{2k+1}^{(R)} &= -(\boldsymbol{e}_{2k-1} + \boldsymbol{e}_{2k} - \boldsymbol{e}_{2k+1} + \boldsymbol{e}_{2k+2}) \otimes A_4
\end{aligned}$$

while each $H_j^{(B)} \in \mathbb{R}^{8p^2,4p}$ is defined by [16]:

$$\begin{aligned}
H_1^{(B)} &= (\boldsymbol{e}_1 + \boldsymbol{e}_2 - \boldsymbol{e}_3) \otimes A_3\\
H_2^{(B)} &= -(\boldsymbol{e}_1 - \boldsymbol{e}_2 + \boldsymbol{e}_3) \otimes A_4\\
H_{2p-1}^{(B)} &= (\boldsymbol{e}_{2p-2} + \boldsymbol{e}_{2p-1} + \boldsymbol{e}_{2p}) \otimes A_3\\
H_{2p}^{(B)} &= -(\boldsymbol{e}_{2p-2} + \boldsymbol{e}_{2p-1} - \boldsymbol{e}_{2p}) \otimes A_4
\end{aligned}$$

and for $k = 2, \ldots, p-2$

$$\begin{aligned}
H_{2k-1}^{(B)} &= (\boldsymbol{e}_{2k-2} + \boldsymbol{e}_{2k-1} + \boldsymbol{e}_{2k} - \boldsymbol{e}_{2k+1}) \otimes A_3\ .\\
H_{2k}^{(B)} &= -(\boldsymbol{e}_{2k-2} + \boldsymbol{e}_{2k-1} - \boldsymbol{e}_{2k} + \boldsymbol{e}_{2k+1}) \otimes A_4
\end{aligned}$$

In all the above, $\boldsymbol{e}_j$ denotes the j-th unit vector in $\mathbb{R}^{2p}$ while the matrices $A_i \in \mathbb{R}^{4p,4p}$ are as defined in [11]. With this formulation of the matrices, it can be shown[16] that for any vector $\boldsymbol{v} \in \mathbb{R}^{8p^2}$, partitioned conformably as

$$v = [\boldsymbol{v}_1^T, \ldots, \boldsymbol{v}_{2p}^T]^T$$

there holds

$$H_R\, \boldsymbol{v} = \sum_{j=1}^{2p} H_j^{(R)} \diamond \boldsymbol{v}_j\ , \tag{8}$$

where

$$H_j^{(R)} \diamond \boldsymbol{v}_j \doteq \begin{cases} (\sum c_i \boldsymbol{e}_i) \otimes (A_4 \boldsymbol{v}_j) & \text{when } j = \text{odd}\\ (\sum \hat{c}_i \boldsymbol{e}_i) \otimes (A_3 \boldsymbol{v}_j) & \text{when } j = \text{even}\end{cases}$$

with the constants c_j and $\hat{c}_j$ to be obviously defined for each j by (7). Similarly,

$$H_B\, \boldsymbol{v} = \sum_{j=1}^{2p} H_j^{(B)} \diamond \boldsymbol{v}_j\ , \tag{9}$$

where

$$H_j^{(B)} \diamond \boldsymbol{v}_j \doteq \begin{cases} (\sum d_i \boldsymbol{e}_i) \otimes (A_3 \boldsymbol{v}_j) & \text{when } j = \text{odd}\\ (\sum \hat{d}_i \boldsymbol{e}_i) \otimes (A_4 \boldsymbol{v}_j) & \text{when } j = \text{even}\end{cases}\ .$$

From relations (8) and (9) it becomes apparent that, in order to efficiently manage a block matrix-vector computation involving H_R (equiv. H_B) in both serial and parallel environments, one has to preprocess the vector by multiplying all of its odd (equiv. even) partitioning blocks by A_4 and all of its even (equiv. odd) partitioning blocks by A_3.

3 MPI Management

In this section, considering the observations made in the previous section, we focus our attention on the MPI

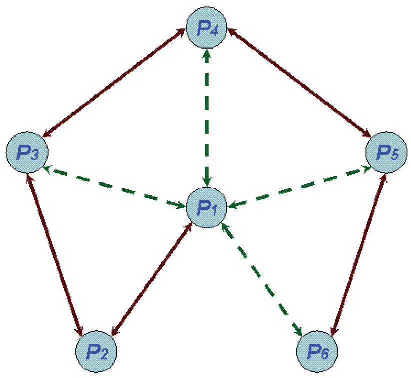

Figure 1 : The Parallel Architecture

management of the computation involved with the B-GS preconditioned Bi-CGSTAB, as the SOR case can be treated in a similar way and has been treated in some detail in [10] on a similar environment. The whole discussion is based on our work in [9, 10] where, taking into consideration the essential factors of (a) uniform load balancing, (b) minimal idle cycles of processors,

Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp290-295)

and (c) minimal communication cost, we partitioned a virtual architecture, in an optimal way for general operators, and mapped it on a proposed (basically pipelined) architecture consisting of $\mathcal{P}_j\ ,\ j=1,\cdots,N$ processors (depicted in Figure 1 for $N=6$). Here, since we deal with operators of Helmholtz-type, we follow a detour to arrive at a more balanced computation. Referring to Figure 1 we remark that:

• Processor $\mathcal{P}_1$, in addition to the computational tasks assign to each processor, has been also assigned the tasks of *gathering* partially processed data, assemble, in the sequel, the final values for the inner products and other parameters of the algorithm, and finally *broadcast* (green dashed communication lines) the results to all other processors.

• Assuming that $k=2p/N$ is an even integer (other cases can be treated similarly), each processor $\mathcal{P}_j$ computes on k *red* and k *black* subvectors of size $4p$. More specifically, to each processor $\mathcal{P}_j$ we assign all the necessary tasks to compute the solution subvectors

$$\boldsymbol{x}_l^{(R)}\ ,\ l=(j-1)k+1,\cdots,jk$$

and

$$\boldsymbol{x}_l^{(B)}\ ,\ l=2p+(j-1)k+1,\cdots,2p+jk\ .$$

• The communication between processors $\mathcal{P}_j$ and $\mathcal{P}_{j+1}$ in order to compute the matrix-vector product $H_R\boldsymbol{z}_R$ is depicted in Figure 2, while for the $H_B\boldsymbol{z}_B$ is depicted in Figure 3.

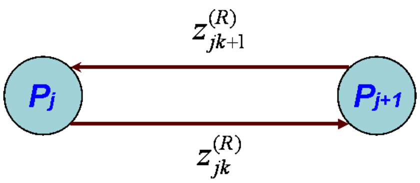

Figure 2 : Communication needed to compute $H_R\boldsymbol{z}_R$

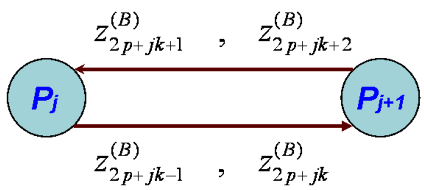

Figure 3 : Communication needed to compute $H_B\boldsymbol{z}_B$

• The total communication between processors, needed in each iteration step of the B-GS preconditioned Bi-CGSTAB, as well as the communication scheduling (for the *black* and the *red* cycles respectively) is being depicted in Figure 4.

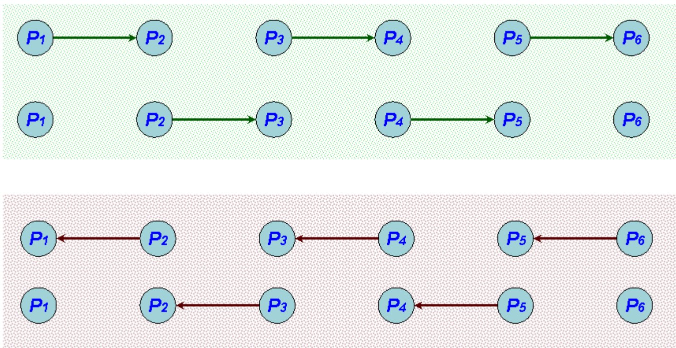

Figure 4 : Communication Scheduling

• In the local memory of each processor, before iteration starts, we store copies of the matrices A_3 and A_4, as well as copies of the LU-factored matrices A_1 and A_2. We also store the appropriate parts of the initial $\boldsymbol{x}^{(0)}$ and the RHS vectors.

Taking into consideration all the above, the program code each processor $\mathcal{P}_j\ ,\ j=1,\cdots,N$ executes for the intense matrix-vector operations (see section 2)

$$\begin{array}{l}\text{Solve}\ \ D_B\,\boldsymbol{z}_B\ =\ \boldsymbol{s}_B\\ \boldsymbol{y}\ =\ H_B\,\boldsymbol{z}_B\\ \text{Solve}\ \ D_R\,\boldsymbol{z}_R\ =\ \boldsymbol{s}_R\ -\ \boldsymbol{y}\\ \boldsymbol{y}\ =\ H_R\,\boldsymbol{z}_R\end{array}$$

takes the specific form:

<u>*Black Cycle*</u>

$$\begin{array}{l}\texttt{do}\quad l=2p+(j-1)k+1\quad\texttt{to}\quad 2p+jk-1\ \ ,\ 2\\ \quad\texttt{Solve}\quad 2A_1\boldsymbol{z}_l^{(B)}=\boldsymbol{s}_l^{(B)}\\ \quad\boldsymbol{y}_{l-2p}=A_3\boldsymbol{z}_l^{(B)}\\ \quad\texttt{Solve}\quad 2A_2\boldsymbol{z}_{l+1}^{(B)}=\boldsymbol{s}_{l+1}^{(B)}\\ \quad\boldsymbol{y}_{l+1-2p}=A_4\boldsymbol{z}_{l+1}^{(B)}\\ \texttt{enddo}\end{array}$$

$$\begin{bmatrix}\boldsymbol{tc}_1\\ \boldsymbol{tc}_2\end{bmatrix}\leftarrow\texttt{Receive}\begin{bmatrix}\boldsymbol{y}_{(j-1)k-1}\\ \boldsymbol{y}_{(j-1)k}\end{bmatrix}\ \texttt{from}\ \ \mathcal{P}_{j-1}$$

$$\texttt{Send to}\ \ \mathcal{P}_{j+1}\ \begin{bmatrix}\boldsymbol{y}_{jk-1}\\ \boldsymbol{y}_{jk}\end{bmatrix}$$

$$\texttt{Send to}\ \ \mathcal{P}_{j-1}\ \begin{bmatrix}\boldsymbol{y}_{(j-1)k+1}\\ \boldsymbol{y}_{(j-1)k+2}\end{bmatrix}$$

$$\begin{bmatrix}\boldsymbol{tc}_3\\ \boldsymbol{tc}_4\end{bmatrix}\leftarrow\texttt{Receive}\begin{bmatrix}\boldsymbol{y}_{jk+1}\\ \boldsymbol{y}_{jk+2}\end{bmatrix}\ \texttt{from}\ \ \mathcal{P}_{j+1}$$

$$\boldsymbol{tm}_1\leftarrow\boldsymbol{tc}_1+\boldsymbol{tc}_2$$

$$\boldsymbol{tm}_2\leftarrow\boldsymbol{y}_{(j-1)k+1}-\boldsymbol{y}_{(j-1)k+2}$$

do $\quad l = (j-1)k+1$ to $jk-3$, 2
$\quad \boldsymbol{tm}_3 \leftarrow \boldsymbol{y}_l$
$\quad \boldsymbol{y}_l \leftarrow \boldsymbol{tm}_2 - \boldsymbol{tm}_1$
$\quad \boldsymbol{tm}_1 \leftarrow \boldsymbol{y}_{l+1} + \boldsymbol{tm}_3$
$\quad \boldsymbol{tm}_2 \leftarrow \boldsymbol{y}_{l+2} - \boldsymbol{y}_{l+3}$
$\quad \boldsymbol{y}_{l+1} \leftarrow \boldsymbol{tm}_1 + \boldsymbol{tm}_2$
enddo
$\boldsymbol{tm}_3 \leftarrow \boldsymbol{y}_{jk-1} + \boldsymbol{y}_{jk}$
$\boldsymbol{y}_{jk-1} \leftarrow \boldsymbol{tm}_2 - \boldsymbol{tm}_1$
$\boldsymbol{y}_{jk} \leftarrow \boldsymbol{tm}_3 + \boldsymbol{tc}_3 - \boldsymbol{tc}_4$

Red Cycle

do $\quad l = (j-1)k+1$ to $jk-1$, 2
$\quad$ Solve $\ 2A_2\boldsymbol{z}_l^{(R)} = \boldsymbol{s}_l^{(R)} - \boldsymbol{y}_l$
$\quad \boldsymbol{y}_l = A_4\boldsymbol{z}_l^{(R)}$
$\quad$ Solve $\ 2A_1\boldsymbol{z}_{l+1}^{(R)} = \boldsymbol{s}_{l+1}^{(R)} - \boldsymbol{y}_{l+1}$
$\quad \boldsymbol{y}_{l+1} = A_3\boldsymbol{z}_{l+1}^{(R)}$
enddo
$\left[\ \boldsymbol{tc}_1\ \right] \leftarrow$Receive $\left[\ \boldsymbol{y}_{(j-1)k}\ \right]$ from $\mathcal{P}_{j-1}$
Send to $\mathcal{P}_{j+1}$ $\left[\ \boldsymbol{y}_{jk}\ \right]$
Send to $\mathcal{P}_{j-1}$ $\left[\ \boldsymbol{y}_{(j-1)k+1}\ \right]$
$\left[\ \boldsymbol{tc}_2\ \right] \leftarrow$ Receive $\left[\ \boldsymbol{y}_{jk+1}\ \right]$ from $\mathcal{P}_{j+1}$
$\boldsymbol{tm}_1 \leftarrow \boldsymbol{tc}_1$
do $\quad l = (j-1)k+1$ to $jk-3$, 2
$\quad \boldsymbol{tm}_2 \leftarrow \boldsymbol{y}_l + \boldsymbol{tm}_1$
$\quad \boldsymbol{tm}_3 \leftarrow \boldsymbol{y}_{l+1} - \boldsymbol{y}_{l+2}$
$\quad \boldsymbol{y}_l \leftarrow \boldsymbol{tm}_2 + \boldsymbol{tm}_3$
$\quad \boldsymbol{tm}_1 \leftarrow \boldsymbol{y}_{l+1}$
$\quad \boldsymbol{y}_{l+1} \leftarrow \boldsymbol{tm}_3 - \boldsymbol{tm}_2$
enddo
$\boldsymbol{tm}_2 \leftarrow \boldsymbol{y}_{jk-1} + \boldsymbol{tm}_1$
$\boldsymbol{tm}_1 \leftarrow \boldsymbol{y}_{jk} - \boldsymbol{tc}_2$
$\boldsymbol{y}_{jk-1} \leftarrow \boldsymbol{tm}_1 + \boldsymbol{tm}_2$
$\boldsymbol{y}_{jk} \leftarrow \boldsymbol{tm}_1 - \boldsymbol{tm}_2$

4 Realization on a DSM computer

SGI Origin 350 is a Shared-Distributed cache coherent - nonuniform memory access (ccNUMA) architecture machine, consisting of eight R16000@600MHz type processors with 4 MB Level 2 cache memory each. The total memory is 4 GB and the operating system is IRIS version 6.5. The applications are developed in double precision Fortran code using the MPI standard for MipsPro compilers version 7.4, which also incorporate the scientific library LAPACK.

The Figures 5 and 6, below, present the speedup measurements of the parallel algorithms, using up to eight processors, for B-GS preconditioned Bi-CGSTAB and SOR methods respectively with discretization of $n_s = 64$, $n_s = 128$ and $n_s = 256$ subintervals.

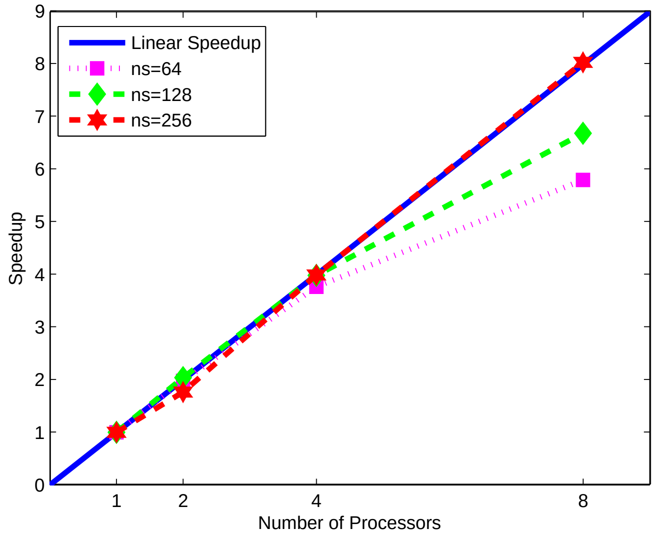

Figure 5 : Speedup measurements for Bi-CGSTAB.

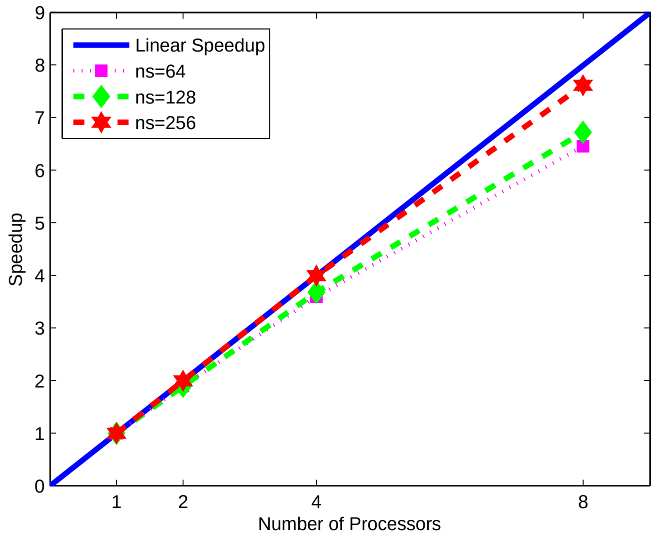

Figure 6 : Speedup measurements for SOR.

As seen in the above figures the speedup is almost linear for up to 4 processors for both methods. For the 8 processors, available in our case, finer discretization yielded almost scalable performance. As a note, we add that the discretization for $n_s = 256$ corresponds to solving a linear system with $262,144$ equations and unknowns.

Acknowledgments :
The authors wish to thank Professor Y.G. Saridakis for the tensor product formulation of the problem and many helpful discussions.

References :

[1] S.H.Brill and G.F.Pinder, Parallel implementation of the Bi-CGSTAB method with Block Red-Black Gauss-Seidel preconditioner applied to the Hermite Collocation discretization of partial differetial equations,*Parallel Computing*, 28, 2002, pp. 399-414.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Toczon and A. White, SourceBook of Parallel Computing, *Morgan Kaufmann Publishers*, 3003.

[3] C.C.Christara,Parallel solvers for spline collocation equations,*Advances in Engineering Software*,27, 1996, pp.71-89,1996.

[4] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald and R. Menon, Parallel Programming in OpenMP, *Academic Press*, 2001.

[5] J.F. Guarnaccia and G.F. Pinder, A Collocation based parallel algorithm to solve immiscible two phase flow in porows media, *Procs of the 5th SIAM conf. on Parallel processing for scientific computing*, 1992, pp. 205-210.

[6] C. E. Houstis, E. N. Houstis and J.R. Rice, Partitioning PDE Computations: Methods and Performance Evaluation, *Parallel Computing*, 5, 1997, pp. 141-163.

[7] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, Mapping Parallel Iterative Algorithms for PDE Computations on a Distributed Memory Computers, *Parallel Alg. and Appl.* , 8,1996, pp. 141-154.

[8] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, Non-Stationary Iterative Schemes for the Solution of Elliptic Collocation Systems, *Procs of the 4nd Hellenic - European conf. on computer mathematics and its applications*, 1998, pp. 1044-1051.

[9] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, Bi-CGSTAB for collocation equations on distributed memory parallel computers , *Numerical Mathematics and advanced applications - ENUMATH 2001*, 2003, pp. 957-966.

[10] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, Iterative Solution of Elliptic Collocation Systems on a Cognitive Parallel Computer, *Computers and Mathematics with applications,* 48, 2004, pp. 951-970.

[11] E. N. Mathioudakis, E. P. Papadopoulou and Y. G. Saridakis, Preconditioning for solving Hermite Collocation by the Bi-CGSTAB, *Procs of the 9th WSEAS Int. Conf. on Applied Mathematics*, 2006, accepted.

[12] Message Passing Interface (MPI) web page, *http://www.mpi-forum.org*

[13] E. P. Papadopoulou, Y. G. Saridakis and T.S. Papatheodorou, Orderings and Partitions of PDE computations for a fixed size VLSI architecture, *Procs of IEEE-ACM Fall Joint Computer Science Conference*, 1997, pp. 366-374.

[14] T.S.Papatheodorou and Y.G.Saridakis, Parallel Algorithms and Architectures for Multisplitting Iterative methods, *Parallel Computing*, 12, 1989, pp. 171-182.

[15] Y. Saad and M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems,*SIAM J. Sci. Statist. Comput.*, 7,1986,pp. 856-869.

[16] Y. G. Saridakis, private communication.

[17] H. A. van der Vorst, Bi-CGSTAB : A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.*, 13,1992, pp. 631-644.