15th Annual Conference of Applied Mathematics, Univ. of Central Oklahoma,
Electronic Journal of Differential Equations, Conference 02, 1999, pp. 75–85.
ISSN: 1072-6691. URL: http://ejde.math.swt.edu or http://ejde.math.unt.edu
ftp ejde.math.swt.edu (login: ftp)

Implementing parallel elliptic solver on a Beowulf cluster *

Marcin Paprzycki, Svetozara Petrova, & Julian Sanchez

Abstract

In a recent paper [5] a parallel direct solver for the linear systems arising from elliptic partial differential equations has been proposed. The aim of this note is to present the initial evaluation of the performance characteristics of this algorithm on Beowulf-type cluster. In this context the performance of PVM and MPI based implementations is compared.

1 Introduction

Recently a new parallel algorithm for the solution of separable second order elliptic PDE's on rectangular domains has been presented by Petrova [5]. The method is based on the sequential algorithm proposed by Vassilevski [7, 8]. The algorithm consists of odd-even block elimination combined with discrete separation of variables. It was established that the proposed solver has good numerical properties for both 2D and 3D problems [5, 7, 8, 9]. The parallel algorithm by Petrova was implemented using PVM to facilitate parallelism and the initial performance evaluation has been reported in [2] (for a brief descriptions of PVM and MPI programming environments, see below). The performance study has run into technical problems. For obvious reasons, one should use parallel computers to solve large problems. On the computers we had access to (Silicon Graphics machines at NCSA in Urbana), this meant that using the batch-queues was required (there is an imposed limit on the size and time allowed for interactive jobs). We have found that for some reason the PVM environment, when invoked from the NQS (NCSA batch job submission system), was relatively unstable (approximately two of every three jobs hang up when the PVM daemons died). In the mean time, while running MPI-based jobs (in the same environment) we have not encountered such problems (see also [4]). We have therefore re-implemented the algorithm using the MPI environment to facilitate parallelism and experimented with it on a Beowulf cluster. Due to the fact that we

* *1991 Mathematics Subject Classifications:* 65F10, 65N20, 65N30.
Key words and phrases: elliptic solvers, separation of variables, parallel computing, problems with sparse right-hand side.

Published November 29, 1999.
The second author was partially supported by the Alexander von Humboldt Foundation and the Bulgarian Ministry for Education, Science and Technology under Grant MM–98#801.

had two versions of the code we were able to run both of them for problems of the same size and compare their performance. The aim of this note is to report on the results of our experiments.

Parallel Virtual Machine (PVM) is a programming environment (developed at Oak Ridge National Laboratory) which allows a heterogeneous collection of workstations and/or supercomputers to function as a single high-performance parallel machine. PVM was designed to link computing resources and provide users with a parallel platform for running their computing applications, irrespective of the number of different computers they use and where the computers are located. It was created primarily as an interactive-type tool, where a console is started on one of the machines and the interactions with other computers are handled from this console. Over the last two years its role has been slowly descreasing. In the meantime, the popularity of the Message Passing Interface (MPI) is increasing. MPI is a message passing library facilitating parallel programming (primarily for codes written in C and Fortran). It was designed in Argonne National Laboratory and released in 1994. In contrast to PVM, MPI is just a specification of a library without an attempt to build an interactive parallel environment. Both PVM and MPI became standards and are supported by most vendors of parallel computers.

We proceed as follows. In Section 2 the method of discrete separation of variables is presented. Section 3 contains a brief summary of the Fast Algorithm for Separation of Variables (FASV) (for more details of both phases of the algorithm, see [5, 7]). In Section 4 we discuss a number of issues related to the parallel implementation and execution. Finally, Section 5 presents the results of experiments performed on a Beowulf cluster. We conclude with the description of future research.

2 Discrete separtion of variables

Consider a separable second order elliptic equation of the form

$$-\sum_{s=1}^{d} \frac{\partial}{\partial x_s} a_s(x_s) \frac{\partial u}{\partial x_s} = f(x), \ x \in \Omega, \quad d = 2 \ \text{ or } 3$$

$$u_{|\partial\Omega} = 0.$$

For the purpose of this paper we assume that Ω is a rectangle ($d = 2$), but the method described below can be generalized to 3D problems. Using, for example, a finite difference method to discretize the equation, we obtain the linear system of equations

$$A\underline{x} = \underline{f}, \tag{2.1}$$

where

$$A = I_m \otimes T + B \otimes I_n.$$

Here I_n is the identity $n \times n$ matrix and $\otimes$ is the *Kronecker product*. Matrices $T = (t_{ij})_{i,j=1}^{n}$ and $B = (b_{kl})_{k,l=1}^{m}$ are tridiagonal s.p.d. arising from the finite

difference approximation of the one-dimensional operators $-\frac{\partial}{\partial x_s}a_s(x_s)\frac{\partial}{\partial x_s}(.)$, for $s = 1, 2$, respectively. Vector $\underline{x}$ and the right-hand side $\underline{f}$ of (2.1) are composed using the lexicographic ordering on horizontal lines.

To describe the method of separation of variables we will use vectors $\underline{x}'$ and $\underline{f}'$ reordered using vertical lexicographic ordering. Consider the following eigenvalue problem

$$B\underline{q}_k = \lambda_k \underline{q}_k, \tag{2.2}$$

where $\left\{\lambda_k, \underline{q}_k\right\}_{k=1}^{m}$ are the eigenpairs of $B_{m\times m}$. The matrix B is assumed s.p.d. and hence the eigenvalues $\lambda_k > 0$ and the eigenvectors satisfy $\underline{q}_k^T \underline{q}_r = \delta_{kr}$ (δ_{kr} is the Kronecker symbol) for $k, r = 1, 2, \ldots, m$. Using the basis $\{\underline{q}_k\}_{k=1}^{m}$ the vectors $\underline{x}'_i$ and $\underline{f}'_i$ can be expanded as follows:

$$\underline{x}'_i = \sum_{k=1}^{m} \eta_{ki}\underline{q}_k, \qquad \underline{f}'_i = \sum_{k=1}^{m} \beta_{ki}\underline{q}_k, \quad i = 1, 2, \ldots, n, \tag{2.3}$$

where the Fourier coefficients of $\underline{f}'_i$ are computed by

$$\beta_{ki} = \underline{q}_k^T \underline{f}'_i. \tag{2.4}$$

Consider the column vectors $\underline{\eta}_k = (\eta_{ki})_{i=1}^{n}$ and $\underline{\beta}_k = (\beta_{ki})_{i=1}^{n}$, $k = 1, 2, \ldots, m$. Substituting expressions (2.3) in (2.1) results in the following system of equations for the discrete Fourier coefficients $\underline{\eta}_k$ of $\underline{x}'_i$, $i = 1, 2, \ldots, n$

$$(\lambda_k I + T)\underline{\eta}_k = \underline{\beta}_k, \quad k = 1, 2, \ldots, m. \tag{2.5}$$

Equations (2.5) represent m systems of linear equations with $n \times n$ matrices. They can be solved independently of each other. The algorithm for the separation of variables (SV) has thus the following form:

Algorithm SV

(1) **determine** the eigenpairs $\left\{\lambda_k, \underline{q}_k\right\}_{k=1}^{m}$ of the tridiagonal matrix B from (2.2);
(2) **compute** the Fourier coefficients β_{ki} of $\underline{f}'_i$ using (2.4);
(3) **solve** m $n\times n$ tridiagonal systems of equations of the form (2.5) to determine $\{\eta_{ki}\}$ – the Fourier coefficients of $\underline{x}'_i$, $i = 1, 2, \ldots, n$;
(4) **recover** the solution of our original system (2.1) on the basis of the Fourier coefficients $\{\eta_{ki}\}$ of $\underline{x}'_i$. There are two possibilities:

$$\begin{aligned} &\text{vertical recovering:} && \underline{x}'_i = \sum_{k=1}^{m} \eta_{ki}\underline{q}_k, \quad i = 1, 2 \ldots, n \\ &\text{horizontal recovering:} && \underline{x}_j = \sum_{k=1}^{m} q_{jk}\underline{\eta}_k, \quad j = 1, 2 \ldots, m. \end{aligned} \tag{2.6}$$

Let us now assume that the system of the form (2.1) has a sparse right-hand side (SHRS) (for more details of origins of such problems see for instance Banegas [1], Proskurowski [6] and Kuznetsov [3]). More precisely, assume that the right-hand side $\underline{f}$ has only $d \ll m$ nonzero block components

$$\underline{f}^T = \left[\underline{0}, \ldots, \underline{f}_{j_1}, \underline{0}, \ldots, \underline{f}_{j_d}, \underline{0}, \ldots, \underline{0}\right]^T, \quad \text{where } \underline{f}_{j_s} \in \mathcal{R}^n, \ s = 1, 2 \ldots, d.$$

Then, for the reordered right-hand side, each vector $\underline{f}'_i$ $(i = 1, 2, \ldots, n)$ has only d nonzero scalar components f_{ij_s}, $s = 1, 2 \ldots, d$, i.e.

$$\underline{f}'^T_i = [0, \ldots, f_{ij_1}, 0, \ldots, f_{ij_d}, 0, \ldots, 0]^T, \quad i = 1, 2, \ldots, n.$$

Assume that only $r \ll m$ block components of the solution are needed. Denote by $\underline{x}_{j'_1}, \underline{x}_{j'_2}, \ldots, \underline{x}_{j'_r}$ the sought vectors. To find the solution of such a problem with a sparse right-hand side we apply the Algorithm SV as follows:

Algorithm SRHS

(1) **compute** the Fourier coefficients β_{ki} of $\underline{f}'_i$ from (2.4),

$$\beta_{ki} = \underline{q}_k^T \underline{f}'_i = \sum_{s=1}^{d} q_{kj_s} f_{ij_s}, \quad k = 1, 2, \ldots, m, \quad i = 1, 2, \ldots, n;$$

(2) **solve** systems of the form (2.5);
(3) **recover** the solution per lines using (2.6). We need only $\underline{x}_j = \sum_{k=1}^{m} q_{jk} \underline{\eta}_k$, for $j = j'_1, j'_2, \ldots, j'_r$.

3 Fast algorithm for separation of variables

Consider now the algorithm FASV proposed by Vassilevski [7, 8] for solving problems of type (2.1). For simplicity we assume that $m = 2^l - 1$. The algorithm consists of two steps – forward and backward recurrence. For the forward step we need matrix A in the following block form:

$$A = \begin{bmatrix} A^{(k,1)} & A_{12} & & & & 0 \\ A_{21} & T + b_{2^k 2^k} I_n & A_{23} & & & \\ & A_{32} & A^{(k,2)} & A_{34} & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & A_{2^{l-k+1}-1, 2^{l-k+1}-2} & A^{(k,2^{l-k})} & \end{bmatrix}, \quad (3.1)$$

where

$$A^{(k,s)} = I_{2^k - 1} \otimes T + B_k^{(s)} \otimes I_n, \quad s = 1, 2, \ldots, 2^{l-k},$$

i.e. each matrix $A^{(k,s)}$, $1 \le k \le l$, $1 \le s \le 2^{l-k}$ has $2^k - 1$ blocks of order n.

Above $B_k^{(s)}$ of order $2^k - 1$ is the principal submatrix of B and hence, it is also a tridiagonal s.p.d. matrix of the form $B_k^{(s)} = (b_{ij})$, $i, j = 1, \ldots s_k + 2^k - 1$, where $s_k = (s-1)2^k$.

(1) **Forward step** of FASV

For $k = 1, 2, \ldots, l-1$ solve the problem:

$$A^{(k,s)}\underline{x}^{(k,s)} = \underline{f}^{(k,s)}, \quad s = 1, 2, \ldots, 2^{l-k}, \tag{3.2}$$

where

$$\underline{x}^{(k,s)} = \begin{bmatrix} \underline{x}^{(k)}_{s_k+1} \\ \underline{x}^{(k)}_{s_k+2} \\ \vdots \\ \underline{x}^{(k)}_{s_k+2^k-1} \end{bmatrix}$$

and $\underline{f}^{(k,s)}$ will be defined below.

After solving these problems and setting $\underline{x}^{(k)}_{s2^k} = 0$ for $s = 1, 2, \ldots, 2^{l-k} - 1$ we denote by $\underline{x}^{(k)}$ and $\underline{f}^{(k)}$ the following vectors

$$\underline{x}^{(k)} = \begin{bmatrix} \underline{x}^{(k,1)} & \} 2^k - 1 \text{ blocks} \\ \underline{0} & \} 1 \text{ block} \\ \underline{x}^{(k,2)} & \} 2^k - 1 \text{ blocks} \\ \underline{0} & \} 1 \text{ block} \\ \vdots & \\ \underline{x}^{(k,2^{l-k})} & \} 2^k - 1 \text{ blocks} \end{bmatrix} \quad \text{and} \quad \underline{f}^{(k)} = \begin{bmatrix} \underline{f}^{(k,1)} \\ \underline{f}_{2^k} \\ \underline{f}^{(k,2)} \\ \vdots \\ \underline{f}^{(k,2^{l-k})} \end{bmatrix}. \tag{3.3}$$

Let the residual vector be the right-hand side for the next $k+1$st step

$$\underline{f}^{(k+1)} = \underline{f}^{(k)} - A\underline{x}^{(k)} = \begin{bmatrix} \underline{f}^{(k+1,1)} \\ \underline{f}_{2^{k+1}} \\ \underline{f}^{(k+1,2)} \\ \vdots \\ \underline{f}^{(k+1,2^{l-k-1})} \end{bmatrix}, \tag{3.4}$$

where

$$\underline{f}^{(k+1,s')} = \begin{bmatrix} \underline{0} & \} 2^k - 1 \text{ blocks} \\ \underline{f}^{(k+1)}_{s'.2^{k+1}} & \} 1 \text{ block} \\ \underline{0} & \} 2^k - 1 \text{ blocks} \end{bmatrix}, \quad s' = 1, 2, \ldots, 2^{l-k-1}.$$

From (3.4) and $s = (2s' - 1)$ we have

$$\begin{aligned} \underline{f}^{(k+1)}_{s'.2^{k+1}} &= \underline{f}^{(k)}_{s.2^k} - A_{2s,2s-1}\,\underline{x}^{(k,s)} - A_{2s,2s+1}\,\underline{x}^{(k,s+1)} \\ &= \underline{f}^{(k)}_{s.2^k} - b_{s.2^k,s.2^k-1}\,\underline{x}^{(k,s)}_{2^k-1} - b_{s.2^k,s.2^k+1}\,\underline{x}^{(k,s+1)}_{1}. \end{aligned} \tag{3.5}$$

The new right-hand side $\underline{f}^{(k+1,s')}$ has only one nonzero block component and hence, by induction, the right-hand sides $\underline{f}^{(k,s)}$ have the following sparsity pattern

$$\underline{f}^{(k,s)} = \begin{bmatrix} \underline{0} \\ * \\ \underline{0} \end{bmatrix} \begin{matrix} \}2^{k-1}-1 \text{ blocks} \\ \}1 \text{ block} \\ \}2^{k-1}-1 \text{ blocks} \end{matrix}, \quad s = 1, 2, \ldots, 2^{l-k}.$$

Therefore, the problems (3.2) have a sparse right-hand side. The matrices $A^{(k,s)}$ allow a separation of variables as submatrices of A and hence, Algorithm SV from Section 2 can be applied with $d = 1$ (the number of nonzero block components of the right-hand side), and $r = 3$ (the number of the sought block components of the solution: $\underline{x}_1^{(k,s)}$, $\underline{x}_{2^{k-1}}^{(k,s)}$ and $\underline{x}_{2^k-1}^{(k,s)}$).

(2) **Backward step** of FASV

For $k = l, l-1, \ldots, 1$, our purpose is to determine $\underline{x}_{(2s-1)2^{k-1}}$ for $s = 1, 2, \ldots, 2^{l-k}$. First, when $k = l$, we solve

$$A\underline{x}^{(l)} = \underline{f}^{(l)}, \tag{3.6}$$

where $A^{(l,s)} \equiv A$, $\underline{x}^{(l)} = \underline{x}^{(l,1)}$ and $\underline{f}^{(l)} = \underline{f}^{(l,1)}$. The right-hand side $\underline{f}^{(l)}$ is found at the last step of the forward recurrence and from (3.5) it has a sparse right-hand side. The problem (3.6) is solved incompletely finding only one block component $\underline{x}_{2^{l-1}}^{(l)}$.

We have also $\underline{x}_{2^{l-1}} = \underline{x}_{2^{l-1}}^{(l)}$ which corresponds to the midblock component of the solution $\underline{x}$ of (2.1). The remaining block components of $\underline{x}$ are recovered by induction as follows:

Starting with $k = l$ we have $\underline{x}_{2^{l-1}} = \underline{x}_{2^{l-1}}^{(l)}$. Assume that for some given k, $1 \le k \le l-1$, we have found the vectors $\underline{x}_{s.2^k}$ for $s = 1, 2, \ldots, 2^{l-k} - 1$ in the previous steps $k+1, \ldots, l$.

Then at the kth step we can find $\underline{x}_{s.2^{k-1}}$ for $s = 1, 2, \ldots, 2^{l-k+1} - 1$. More precisely, by construction we have $\underline{f}^{(k'+1)} = \underline{f}^{(k')} - A\underline{x}^{(k')}$ and after summation of both sides of these equalities for $k' = k, k+1, \ldots, l$, we get

$$\underline{f}^{(k)} = A\left(\sum_{k'=k}^{l} \underline{x}^{(k')}\right). \tag{3.7}$$

Let

$$\underline{y} = \sum_{k'=k}^{l} \underline{x}^{(k')} \tag{3.8}$$

and using A from (3.1) we have the following equivalent form for (3.7)

$$\begin{bmatrix} A^{(k,1)} & A_{12} & & & & 0 \\ A_{21} & T+b_{2^k2^k}I_n & A_{23} & & & \\ & A_{32} & A^{(k,2)} & A_{34} & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & A_{2^{l-k+1}-1,2^{l-k+1}-2} & A^{(k,2^{l-k})} & \end{bmatrix} \begin{bmatrix} \underline{y}^{(k,1)} \\ \underline{y}_{2^k} \\ \underline{y}^{(k,2)} \\ \vdots \\ \underline{y}^{(k,2^{l-k})} \end{bmatrix}$$

$$= \begin{bmatrix} \underline{f}^{(k,1)} \\ \underline{f}_{2^k} \\ \underline{f}^{(k,2)} \\ \vdots \\ \underline{f}^{(k,2^{l-k})} \end{bmatrix}$$

where each block $\underline{y}^{(k,s)},\ s=1,2,\ldots,2^{l-k}$ has 2^k-1 blocks of order n. From this system we obtain the following equation for $\underline{y}^{(k,s)}$:

$$A_{2s-1,2s-2}\underline{y}_{(s-1).2^k} + A^{(k,s)}\underline{y}^{(k,s)} + A_{2s-1,2s}\underline{y}_{s.2^k} = \underline{f}^{(k,s)}. \tag{3.9}$$

Hence, when $s_k=(s-1)2^k$:

$$A^{(k,s)}\underline{y}^{(k,s)} = \underline{f}^{(k,s)} - A_{2s-1,2s-2}\underline{y}_{s_k} - A_{2s-1,2s}\underline{y}_{s.2^k}. \tag{3.10}$$

Using (3.8) we have $\underline{y}_{s_k} = \underline{x}_{s_k}$ and $\underline{y}_{s.2^k} = \underline{x}_{s.2^k}$. Recall that from the induction described above, the vectors $\underline{x}_{s_k} = \underline{x}_{(s-1).2^k}$ and $\underline{x}_{s.2^k}$ are already found. Thus, from (3.10) we get the following system

$$A^{(k,s)}\underline{y}^{(k,s)} = \begin{bmatrix} -b_{s_k+1,s_k}\underline{x}_{s_k} \\ \underline{0} \\ \underline{f}^{(k,s)}_{2^{k-1}} \\ \underline{0} \\ -b_{s.2^k-1,s.2^k}\underline{x}_{s.2^k} \end{bmatrix} \begin{matrix} \}1 \text{ st block} \\ \\ \}2^{k-1} \text{ th block} \\ \\ \}2^k-1 \text{ st block} \end{matrix} . \tag{3.11}$$

The nonzero block components of the right-hand side of (3.11) can be found using the solution computed at the $k+1$st step. It is a problem with a sparse right-hand side and only one ($r=1$) block component of the solution, namely $\underline{y}^{(k,s)}_{2^{k-1}}$ is needed. By (3.8) one finds

$$\underline{y}^{(k,s)}_{2^{k-1}} = \underline{y}_{(2s-1).2^{k-1}} = \sum_{k'=k}^{l} \underline{x}^{(k')}_{(2s-1).2^{k-1}} = \underline{x}_{(2s-1).2^{k-1}}.$$

Therefore, at the kth step from the backward recurrence ($k=l,\ l-1,\ldots,1$) we can determine $\underline{x}_{(2s-1)2^{k-1}},\ s=1,2,\ldots,2^{l-k}$.

4 Parallel implementation

When a 2D problem is solved the rectangular domain is decomposed into horizontal strips. We then assign a processor to each strip. In each forward step of FASV a number of solves with matrices $A^{(k,s)}$ are involved. These are problems with sparse right hand sides and to find the components of the solution algorithm SRHS is applied. In general, the forward sweep of FASV requires $O(log(m))$ steps. The backward sweep of FASV is a reverse of the forward step and results in the solution of the original system (2.1).

In [5] it was shown that the total arithmetical complexity of the algorithm is $(28n^2 - 9n^2/(l-1))(l-3-logP+2P)/P$ and the speed-up $S = (l-1)P/(l-3-logP+2P)$. Therefore, in the two limiting cases, for large l the optimal speed-up P is obtained, while for a fixed l and a large P the speed-up is limited by $l/2$.

As mentioned above, we have used two versions of the code. The only difference between them was that in one the PVM environment was used to facilitate interprocessor communication, while the other was based on the MPI library. While re-implementing the code we have fixed the way that the time was measured. In the original code an average of processor times was reported. We have decided that this may be slightly misleading as it hides possible workload imbalances. In the new version of the code we measured the time spent by each individual processor. Since the workload differs from processor to processor, in each run we recorded the longest time (all remaining processors have to wait for the slowest one to complete its job before the problem is solved). After performing several runs, we kept and reported the shortest time (of the longest times). We used the function $mclock()$ to measure time in the PVM version while in the MPI version we used the $MPI\_Wtime()$ function.

Our experiments have been performed on a Beowulf cluster. The Beowulf cluster architecture was designed in 1994 by Thomas Sterling and Donald Beck at the center of Excellence in Space Data and Information Sciences (CESDIS). The purpose of the design was to combine together COTS (commodity off the shelf) base systems in a network and emulate a costly high performance supercomputer.

The Beowulf cluster at the University of Southern Mississippi consists of 16 PC's connected with a Fast Ethernet switch. Each PC is a 233 MHz Pentium II with 256 Mbytes of RAM. The proposed algorithm has been implemented in Fortran (77). We used the Portland Group compiler and invoked maximum compiler optimization ($-O2\ -tp\ p6\ -Mvect\ -Munroll = n : 16$). Most runs have been done on an empty machine.

5 Experimental results

To study the performance of the codes we have used a standard model problem. We have solved the Poisson equation on the unit square with Dirichlet boundary conditions. The mesh size was fixed at $h = 1/(m+1)$, where $m = 2^l$. In Table 1

P	$L = 9$		$L = 10$		$L = 11$	
	MPI	PVM	MPI	PVM	MPI	PVM
1	1.16	1.30	6.32	6.71	32.19	34.42
2	0.69	0.75	3.50	3.91	17.50	19.20
3	0.52	0.58	2.54	2.78	13.39	13.50
4	0.42	0.46	2.01	2.30	9.68	10.92
5	0.40	0.40	1.81	1.95	8.63	9.23
6	0.38	0.41	1.68	1.91	7.77	8.84
7	0.36	0.33	1.53	1.72	6.94	7.88
8	0.32	0.32	1.36	1.54	6.15	7.00
9	0.34	0.30	1.35	1.43	6.02	6.05
10	0.35	0.32	1.32	1.48	5.78	5.53
11	0.34	0.30	1.28	1.38	5.48	5.77

Table 1: Times comparison

we present the times obtained for both PVM and MPI codes for $l = 9, 10, 11$ and for $P = 1, 2, \ldots, 11$ processors (unfortunately, due to the technical difficulties 5 nodes were down for hardware maintenance). Problem of size $l = 11$ was the largest that we were able to run on a single processor system (it required more than 90Mbytes of memory).

It can be observed that for the larger problems the MPI based code slightly outperformed the PVM based implementation. This result is consistent with our expectations. One needs to remember that MPI is a newer development than PVM and performance was more of a goal in its creation (in case of PVM the main goal was rather ability of interactively creating a heterogeneous parallel computer system). Additionally, the Beowulf implementation of MPI, which uses the LAM communication fabrics, supports sending and receiving messages directly from one process to another, overriding the communication daemons. LAM is a parallel processing environment and development system for a network of independent computers created and maintained by the Ohio Supercomputer Center. It features the MPI programming standard, supported by extensive monitoring and debugging tools.

In Table 2 we compare the speed-up values of both implementations for the largest problem size. In addition we present the theoretical speed-up value calculated using the speed-up formula above.

Interestingly, both MPI and PVM speedup values are far superior to the theoretical ones. This shows a slight weakness of the theoretical arithmetical complexity analysis which typically takes into consideration only the size of the problem (sometimes supplemented by a rudimentary data communication analysis). In the experimental environment, one deals with machines with hierarchical memory structure and its management as well as networks and network protocols which influence the performance of the program. For instance, as we increase the number of processors for a given fixed problem size the amount of

P	$L = 11$		
	MPI	PVM	THEORY
1	1.00	1.00	1.00
2	1.84	1.79	1.67
3	2.40	2.55	2.14
4	3.33	3.15	2.50
5	3.73	3.73	2.78
6	4.14	3.89	3.00
7	4.64	4.37	3.18
8	5.23	4.92	3.33
9	5.35	5.69	3.46
10	5.57	6.22	3.57
11	5.87	5.96	3.67

Table 2: Speedup comparison

memory used per-processor decreases (since the problem is divided into smaller pieces) and it is much easier to manage the data movement in the hierarchical environment making the multiprocessor runs faster than the predicted ones. This can be viewed also from the reverse side. For very large problems the memory management becomes rather complicated. It is well known that most of the existing cache management algorithms are far from optimal. This increases the single processor execution time and thus increases the obtainable speed-up.

In this context we can also observe that in many cases the PVM based code shows slightly faster speed-up values than MPI. The reason for that behavior is that speed-up formula involves the time spent by a single processor run. Inspecting the values in Table 1 we can see that the single processor PVM version is slower than the MPI version. Thus any speedup value will have a greater numerator and therefore a greater value.

6 Concluding remarks

In this note we have presented the experimental results illustrating the parallel performance characteristics of the fast direct elliptic solver on a Beowulf cluster. The algorithm and both its implementations turned out to be relatively efficient. We have also found that, for large problems, memory management capacity is an important factor influencing performance. In the near future we plan to, first, investigate the code to see if there is a possibility of any additional optimization of the way that the code is currently implemented. Second, we plan to complete the performance study across a number of modern parallel programming architectures. We will also look into extending the proposed approach to parallel three dimensional solvers.

References

[1] A. Banegas, Fast Poisson solvers for problems with sparsity, *Math.Comp.*, **32**(1978), 441-446.

[2] H. Hope, M. Paprzycki and S. Petrova, Parallel Performance of a Direct Elliptic Solver, in: M. Griebel et. al. (eds.), *Large Scale Scientific Computations of Engineering and Environmental Problems*, Vieweg, Wisbaden, (1998), 310-318

[3] Y. Kuznetsov, Block relaxation methods in subspaces, their optimization and application, *Soviet.J.Numer.Anal. and Math. Modelling*, **4**(1989), 433-452.

[4] I. Lirkov, S. Margenov and M. Paprzycki, Benchmarking performance of parallel computers using a 2D elliptic solver, *Proceedings of the* 4^{th} *International Conference on Numerical Methods and Applications*, Sofia, Bulgaria, August, 1998, to appear.

[5] S. Petrova, Parallel implementation of fast elliptic solver, *Parallel Computing*, **23**(1997), 1113-1128.

[6] W. Proskurowski, Numerical solution of Helmholtz equation by implicit capacitance matrix method, *ACM Trans. Math. Software*, **5**(1979), 36-49.

[7] P. Vassilevski, Fast algorithm for solving a linear algebraic problem with separation of variables, *Comptes Rendus de l'Academie Bulgare des Sciences*, **37**(1984), No.3, 305-308.

[8] P. Vassilevski, Fast algorithm for solving discrete Poisson equation in a rectangle, *Comptes Rendus de l'Academie Bulgare des Sciences*, **38**(1985), No.10, 1311-1314.

[9] P. Vassilevski and S. Petrova, A note on construction of preconditioners in solving 3D elliptic problems by substructuring, *Comptes Rendus de l'Academie Bulgare des Sciences*, **41**(1988), No.7, 33-36.

Marcin Paprzycki (e-mail: marcin.paprzycki@usm.edu)
Julian Sanchez (e-mail: julian.sanchez@ieee.org)
Department of Computer Science and Statistics
University of Southern Mississippi
Hattiesburg, MS 39406, USA

Svetozara Petrova
Central Laboratory of Parallel Processing
Bulgarian Academy of Sciences, Acad. G.Bonchev Str.
Block 25A, 1113 Sofia, Bulgaria
e-mail: zara@cantor.bas.bg