Parallel Performance Comparison of Three Direct Separable Elliptic Solvers

Gergana Bencheva *
Central Laboratory for Parallel Processing,
Bulgarian Academy of Sciences
gery@parallel.bas.bg

April 17, 2003

Abstract

The parallel properties of three fast direct solution methods for linear systems with separable block tridiagonal matrices and a related C/MPI code are studied. Fast algorithm for separation of variables and two variants of the generalized marching algorithm are first summarized. The results from numerical tests performed on two coarse-grained parallel architectures are then reported. The obtained speed-up and efficiency coefficients are compared. The general conclusion is that not always the best sequential solver has the best parallel performance.

1 Introduction

A measure for the efficiency of a given sequential direct solver is its computational complexity. The speed-up and efficiency coefficients play a key role in the analysis of parallel algorithms. They are based on the times per processor for computations $T_{comp} = \mathcal{N} * t_a$ and communications $T_{com} = c_1 * t_s + c_2 * t_w$. Here, $\mathcal{N}$ is the number of operations, c_1 characterizes the number of stages at which communications are needed and c_2 is the amount of the transferred words. Both c_1 and c_2 can be constants or functions of the number of processors and/or the size of the problem. The parameters

*This research has been supported in part by the USA National Science Foundation under Grant DMS 9973328, by the Bulgarian Ministry of Education and Science under Grant MU-I-901/99 and by the Center of Excellence BIS-21 Grant ICA1-2000-70016

t_a, t_s and t_w depend on the parallel computer. The largest one of them is t_s and it could be hundreds and even thousands times larger than t_w. That is why one and the same parallel solver may have different behavior on machines with different characteristics.

Our goal was to compare the performance of three direct solvers on two of the coarse-grained parallel architectures available in Texas A&M University. We summarize in this report results obtained on SGI Origin 2000 and Beowulf cluster. Parallelizations of fast algorithm for separation of variables (FSV), and two variants of generalized marching (GM) algorithm - GMF and GMS, based respectively on FSV and on incomplete solution technique for problems with sparse right-hand sides (SRHS) instead of FSV, are considered. The algorithm FSV is proposed in [14]. Its parallelization aspects for Poisson equation on nonuniform mesh are analyzed in [12]. Here we consider slightly different variant proposed in [6]. The GM algorithm is first developed in [2, 3] and later reformulated in [13] using SRHS and FSV. The algorithm GMS is introduced in [7], where the parallel properties of both GMF and GMS are theoretically studied.

In the rest part of the report, we first present briefly the algorithms (Section 2) and their parallel implementation (Section 3). In Section 4 are compared not only the obtained speed-up and efficiency coefficients, but also the measured cpu-times. The general conclusion is that although GMS is the slowest sequential algorithm, its parallel implementation PGMS has better properties and on some machines it is asymptotically the best one among the considered solvers.

2 Fast Separable Solvers

We start the exposition in this Section with formulation of the problem. After that the technique for incomplete solution of problems with sparse right–hand sides (SRHS) originated in Banegas [1], Proskurowski [11] and Kuznetsov [9] is described, since it is a principal step used at different stages in each of the studied solvers. More theoretical aspects of the problems with sparsity are investigated by Kuznetsov in [8]. We next present briefly the fast algorithm for separation of variables (FSV) and the generalized marching (GM) algorithm originated in Bank and Rose [3] and Bank [2]. The two considered variants of GM, denoted for brevity GMF and GMS are based on FSV and SRHS respectively. All of the studied solvers take advantage of the special separable sparse structure of the stiffness matrix.

2.1 Formulation of the problem

A separable second order elliptic equation with non-constant coefficients is considered:

$$\left| \begin{array}{rcll} -\sum_{s=1}^{2} \frac{\partial}{\partial x_s} \left(a_s(x_s) \frac{\partial u}{\partial x_s} \right) & = & f(x), & x = (x_1, x_2) \in \Omega = (0,1)^2 \\ u & = & 0, & \text{on } \partial\Omega \end{array} \right. . \tag{1}$$

It is discretized on rectangular $n \times m$ grid by finite differences or by piece-wise linear finite elements on right–angled triangles. Using the identity $n \times n$ matrix I_n, the tridiagonal, symmetric and positive definite matrices $T = (t_{i,j})_{i,j=1}^{n}$ and $B = (b_{i,j})_{i,j=1}^{m}$, and the Kronecker product $C_{m_1 \times n_1} \otimes D_{m_2 \times n_2} = (c_{i,j} D)_{i=1\, j=1}^{m_1\, n_1}$, $C = (c_{i,j})_{i=1\, j=1}^{m_1\, n_1}$, the obtained discrete problem with $N = n\, m$ degrees of freedom is written in the form:

$$A\mathbf{x} \equiv (B \otimes I_n + I_m \otimes T)\mathbf{x} = \mathbf{f}. \tag{2}$$

Here $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)^T$, $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m)^T$, $\mathbf{x}_j, \mathbf{f}_j \in \mathbf{R}^n$, $j = 1, \ldots, m$ and

$$A \equiv \begin{pmatrix} T + b_{1,1}\, I_n & b_{1,2}\, I_n & & 0 \\ b_{2,1}\, I_n & T + b_{2,2}\, I_n & & 0 \\ \ddots & \ddots & & \vdots \\ 0 & \ldots & b_{m,m-1}\, I_n & T + b_{m,m}\, I_n \end{pmatrix}.$$

Remark: When the boundary conditions are nonzero, i.e. $u = \mu(x)$, on $\partial\Omega$, the resulting system has the same form, but with slightly modified components of $\mathbf{f}$.

2.2 Incomplete Solution Technique

The technique for incomplete solution of problems with sparse right–hand sides is proposed independently by Banegas [1], Proskurowski [11] and Kuznetsov [9]. It is assumed (for a reason to become clear later on) that the right–hand side $\mathbf{f}$ of the system (2) has only d ($d \ll m$) nonzero block components and only r ($r \ll m$) block entries of the solution are needed. Let for definiteness $\mathbf{f}_j = 0$ for $j \neq j_1, j_2, \ldots, j_d$. To find the needed components $\mathbf{x}_{j_1'}, \mathbf{x}_{j_2'}, \ldots, \mathbf{x}_{j_r'}$ of the solution, the well-known algorithm for discrete separation of variables is applied taking advantage of the right–hand side sparsity:

Algorithm SRHS

Step **0.** *determine* all the eigenvalues $\{\lambda_k\}_{k=1}^{m}$ and the needed $\tilde{d} \leq r + d$ entries, $\{\mathbf{q}_{k,j}\}$, $j = j_1, \ , \ldots, \ j_d$, and $j = j_1^{'}, \ \ldots, \ j_r^{'}$, of all the eigenvectors $\{\mathbf{q}_k\}_{k=1}^{m}$ of the tridiagonal matrix B ;

Step **1.** *compute* the Fourier coefficients $\beta_{i,k}$ of $\mathbf{f}_i'$ from the equations:

$$\beta_{i,k} \;=\; \mathbf{q}_k^T \cdot \mathbf{f}'_i \;=\; \sum_{s=1}^{d} q_{j_s,k} f_{i,j_s},\; i = 1,\ldots,n,\; k = 1,\ldots,m\,;$$

Step **2.** *solve* m $n \times n$ independent tridiagonal systems of linear equations:

$$(\lambda_k\, I_n + T)\, \eta_k = \beta_k,\; k = 1,\ldots,m\,;$$

Step **3.** *recover* r components of the solution per lines using

$$\mathbf{x}_j \;=\; \sum_{k=1}^{m} q_{j,k}\eta_k\,,\; j = j'_1, j'_2, \ldots, j'_r$$

END {SRHS}.

As it is shown in [14], the ALGORITHM SRHS requires $\mathcal{N}_{SRHS} \approx 2(r+d)nm+5nm$ arithmetic operations (ar. ops.) in the solution part and $\mathcal{O}(\tilde{d}\, m^2) + 9m^2$ ar. ops. for the computation of all the eigenvalues and the specified $\tilde{d}$ entries of all the eigenvectors of the tridiagonal matrix B.

2.3 Fast Algorithm for Separation of Variables

The Algorithm FSV is proposed in [14]. A detailed description may be found in [10], see also [4]. It consists of forward (FR) and backward (BR) recurrence. Let for simplicity $m = 2^l - 1, l \in \mathbf{Z}$. At each step k of FR and BR, systems with specific sparse right–hand sides are constructed and solved incompletely using ALGORITHM SRHS. The compact form of FSV is:

ALGORITHM FSV

Step **1.** Forward recurrence:
Set $\mathbf{f}^{(1)} = \mathbf{f}$
for $k = 1$ to $l-1$
for $s = 1$ to 2^{l-k} *solve* $A^{(k,s)}\mathbf{x}^{(k,s)} = \mathbf{f}^{(k,s)}$
incompletely, finding only $\mathbf{x}_1^{(k,s)},\; \mathbf{x}_{2^{k-1}}^{(k,s)},\; \mathbf{x}_{2^k-1}^{(k,s)}$
end {loop on s}
for $s = 1$ to $2^{l-k} - 1$ *compute*
$\mathbf{f}_{s2^k}^{(k+1)} = \mathbf{f}_{s2^k}^{(k)} - b_{s2^k,s2^k-1}\mathbf{x}_{2^k-1}^{(k,s)} - b_{s2^k,s2^k+1}\mathbf{x}_1^{(k,s+1)}$
end {loop on s }
end {loop on k}

Step **2.** Backward recurrence:
solve incompletely $A\mathbf{x}^{(l)} = \mathbf{f}^{(l)}$ only for $\mathbf{x}_{2^{l-1}}^{(l)} = \mathbf{x}_{2^{l-1}}$
for $k = l-1$ down to 1
for $s = 1$ to 2^{l-k} *solve* incompletely $A^{(k,s)}\hat{\mathbf{y}}^{(k)} = \hat{\mathbf{f}}^{(k,s)}$
only for $\hat{\mathbf{y}}_{2^{k-1}}^{(k)}$
Then set $\mathbf{x}_{(2s-1)2^{k-1}} = \hat{\mathbf{y}}_{2^{k-1}}^{(k)} + \mathbf{x}_{2^{k-1}}^{(k,s)}$
end {loop on s}
end {loop on k}

END {FSV}.

The matrices $A^{(k,s)}$ consist of 2^k-1 blocks of order n and have the form $A^{(k,s)} = I_{2^{k-1}} \otimes T + B_k^{(s)} \otimes I_n$, $s = 1, 2, \ldots, 2^{l-k}$. They are constructed using the principal submatrices $B_k^{(s)} = \text{tridiag}\{b_{s_k+i,s_k+i-1}, b_{s_k+i,s_k+i}, b_{s_k+i,s_k+i+1}\}_{i=1}^{2^k-1}$ of B for $s_k = (s-1)2^k$. The right–hand sides for the FR have one nonzero component (with index 2^{k-1}) and for the BR they are two such components (with indices 1 and 2^k-1). It is important to note here that the update of the right–hand sides $\mathbf{f}_{s2^k}^{(k+1)}$ for each k requires components of the solution of the subsystems for the previous values of k. The solution of the original system is consecutively recovered during the BR, based on the data determined at previous stages.

2.4 Generalized Marching Algorithm

The GM algorithm is first proposed in [2, 3] and later reformulated in [13] using the incomplete solution technique and ALGORITHM FSV. The standard marching algorithm (ALGORITHM SM) is optimal in the sense that its computational cost $\mathcal{N}_{SM} \approx 31nm$ depends linearly on the size of the discrete system. Unfortunately it is unstable for large m (see [2, 3]) and hence is of practical interest only for $m \ll n$. The GM algorithm is a stabilized version of ALGORITHM SM. We assume that $m+1 = p(k+1)$, for some integers p, k. The original system (2) is first reordered and rewritten into two-by-two block form $\widetilde{A}$ with diagonal blocks $\widetilde{A}_{i,i}$ on the main diagonal. Applying block-Gaussian elimination, it is reduced to solution of two systems with $\widetilde{A}_{1,1}$ and one system with the Schur complement. Here $\widetilde{A}_{1,1} = \mathit{blockdiag}(A_s^{(k)})_{s=1}^{p}$, $A_s^{(k)} = I_k \otimes T + B_s^{(k)} \otimes I_n$, where $B_s^{(k)} = \mathit{tridiag}(b_{k_s+i,k_s+i-1}, b_{k_s+i,k_s+i}, b_{k_s+i,k_s+i+1})_{i=1}^{k}$, $k_s = (s-1)(k+1), s = 1, \ldots, p$. Hence, each of the systems with $\widetilde{A}_{1,1}$ is equivalent to solution of p independent subproblems with matrices $A_s^{(k)}$, $s = 1, \ldots, p$. The stability of SM at this stage is ensured by choosing sufficiently large p.

The second step of GM, i.e. the system with the Schur complement, is equivalent to solution of a system with the original matrix A and with a sparse right-hand side. The algorithm GM is summarized as follows:

ALGORITHM GM

Step **1.** for $s = 1$ to p

solve $A_s^{(k)}\, \mathbf{y}_s^{(1)} = \mathbf{f}_s^{(1)}$ using ALGORITHM SM

end {loop on s}

compute $\widetilde{\mathbf{f}}^{(2)} = \mathbf{f}^{(2)} - \widetilde{A}_{2,1}\, \mathbf{y}^{(1)}$;

Step **2.** *solve* incompletely $A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}$, $\widehat{\mathbf{f}}_i \neq 0$ for $i = s(k+1)$, $(\widehat{\mathbf{f}}_i = \widetilde{\mathbf{f}}_s^{(2)})$

seeking only $\widehat{\mathbf{x}}_{s(k+1)} = \mathbf{x}_{s(k+1)}$, $s = 1, \ldots, p-1$;

Step **3.** *compute* $\widetilde{\mathbf{f}}^{(1)} = \mathbf{f}^{(1)} - \widetilde{A}_{1,2}\, \mathbf{x}^{(2)}$;

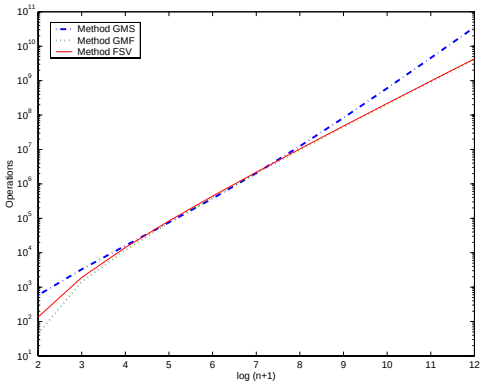

Figure 1: Computational complexity of FSV, GMF and GMS, $k = 7$

for $s = 1$ to p
solve $A_s^{(k)}\,\mathbf{x}_s^{(1)} = \widetilde{\mathbf{f}}_s^{(1)}$ using Algorithm SM
end {loop on s}
End {GM}.

Step 2 of Algorithm GM, as it was proposed in [13], is handled by Algorithm FSV. To find the required components of $\mathbf{x}$ only lp steps of Algorithm FSV are applied - the last lp steps of FR and the first of BR. We refer to this variant of GM as Algorithm GMF. It is asymptotically slightly faster than FSV. If we solve the system at Step 2 simply using Algorithm SRHS, we will get more expensive sequential solver referred as Algorithm GMS. This variant was proposed in [7] to improve the parallelization properties of the GM algorithm. The advantages of GMS in this respect are shown in the remaining part of the present report.

2.5 Computational Complexity

The above described solvers require the following amount of arithmetic operations:

Algorithm FSV $\quad \mathcal{N}_{FSV} \approx 24nm(\log m - 1) - 9nm$

Algorithm GMF $\quad \mathcal{N}_{GMF} \approx 62pkn + 24nm(\log p - 1) - 9nm$

Algorithm GMS $\quad \mathcal{N}_{GMS} \approx 62pkn + 4pnm + nm$

In Figure 1 they are plotted in logarithmic scale for the case $k = 7$. Algorithm GMF is always slightly faster than Algorithm FSV. For some sizes of the discrete problem ($4 < l < 8$, $m = 2^l - 1$) Algorithm GMS is faster than both FSV and GMF, but asymptotically it is the slowest one.

Unfortunately, the variable size of the subsystems solved incompletely in FSV lead to some difficulties in the parallelization of both FSV and GMF. They are discussed in the next Section together with the advantages of GMS.

3 Parallel Implementation

How are the data and the computations distributed among the processors? What are the type and the amount of communications inspired by this division? How good is the developed solver? These are the questions treated in this Section for each of our methods. The parallelizations of the algorithms FSV, GMF and GMS are referred as PFSV, PGMF and PGMS respectively. More detailed description of these parallel solvers together with a comparison of their advantages and disadvantages may be found in [6, 7].

3.1 Initial Data in Each Processor

Let us have $NP = 2^{np}$ processors enumerated from 0 to $NP - 1$. Let us also assume that $m + 1 = 2^l = p(k + 1)$, $p = 2^{lp}$, $k + 1 = 2^{lk}$, $l = lk + lp$, (l, lp, $lk \in \mathbf{Z}$). For FSV and both variants of GM, the computational domain is decomposed into number of strips equal to the number of processors. Their length is $LSTRIP = 2^{l-np}$ except the last one, which length is $LSTRIP - 1$. The nature of the considered solvers, in particular the size and the structure of the systems solved incompletely, requires the data related to these strips to be duplicated in some of the processors. It is done in two different ways illustrated in Figure 2 - one for PGMS and one for PFSV and PGMF. Algorithm PGMS requires the data to be duplicated only in the master

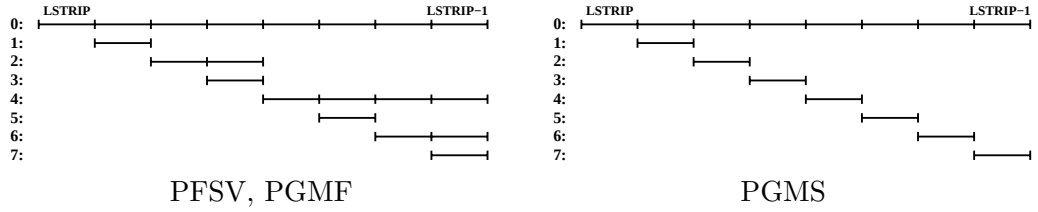

Figure 2: Initial data in each processor, P=8

processor 0, while for the remaining algorithms it should also be done in a specific manner in all even numbered processors.

So the initial data is generated as follows: Each processor contains the whole matrix T and the elements of the matrix B corresponding to one or more successive strips (see Fig. 2). For PFSV and PGMF also parts of the right–hand side have to

be duplicated, while for PGMS it is enough that each processor has the components of **f** only for one strip. The first advantage of PGMS is the less required memory for the initial data.

3.2 Parallel Implementation of FSV

Both FR and BR of PFSV are divided into two stages. First one corresponds to the values of k for which sequential SRHS is used. The second one is for the case when the size of the subsystems solved incompletely is larger than $LSTRIP$ and a parallel implementation of SRHS (PSRHS) should be used to improve the load balance of the computer system. This means that each of the loops on k (see ALGORITHM FSV) should be divided into two parts, namely $k = 1, \ldots, l - np$ and $k = l - np + 1, \ldots, l-1$ for FR and in the reverse order for BR. All the processors have to complete equal amount of computations since for a given k in fact they have to solve 2^{l-k-np} subsystems with SRHS in the first stage and one subsystem with PSRHS (described in the next Section) for the second one.

For the first stage of FR the communications are local – one component of the solution and one component of the right–hand side have to be sent from the processor $myid$ to $myid-1$. For the second stage the same data have to be transferred, but now between processors which are not neighbors. Additional global communications are required in PSRHS. For the case $k = l, \ldots, l - np + 1$ of BR, except communications for PSRHS, only one component of the solution should be transferred, but to all processors which will need it. For the other case no communications are required.

3.3 Parallel Implementation of SRHS

Parallelization of ALGORITHM SRHS is required in all of the considered solvers. In PGMS it is called only once, while in PFSV and PGMF it is done for each k of the second stage of FR and BR. The input and the output data of PSRHS is handled in two slightly different ways – one for PGMS and one for the remaining solvers. Hence some additional communications are to be performed in the second case. The version of PSRHS, schematically presented below, is used in PGMS.

ALGORITHM PSRHS

Step **0.** *compute* $\{\mathbf{q}_k, \lambda_k\}$ (preprocessing step);

Step **1.** *compute* $\beta_{i,k}$ $i = 1, \ldots, n$, $k = 1, \ldots, m$,

$$\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \end{array}\right) \left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right) = \left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \end{array}\right);$$

$\qquad\qquad F_{n\times d} \qquad\qquad Q_{d\times m} \qquad\qquad Beta_{n\times m}$

Step **2.** *solve* $(\lambda_k\, I_n + T)\,\eta_k = \beta_k,\ k = 1, \ldots, m/NP$;

Step **3.** *recover* $\mathbf{x}_j,\ j = j'_1, j'_2, \ldots, j'_r$,

$$\mathbf{x}_j = \left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \end{array}\right) \left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right).$$

$\qquad\qquad\qquad\qquad\qquad\qquad Q_{m\times m} \qquad\qquad V_m$

End {PSRHS}.

In the preprocessing step the master solves the eigenproblem and distributes the data in the required way. Computations and communications at Step 1 are the same as if we have to perform matrix × matrix multiplication. The data for this multiplication is distributed as follows: Left matrix is divided in strips by columns, right - by rows, and the product should be distributed among processors again in strips by columns. This multiplication requires global collective communications (*scatter* and *reduce* = *gather* + *broadcast*). Computations are divided into equal parts. For the next Step 2 each processor have to solve m/NP independent tridiagonal systems without any communications. The third step is performed in similar way as Step 1 with the only difference that the second matrix and the result are now column vectors.

3.4 Parallel Implementation of GM

Both algorithms PGMF and PGMS may be presented in a compact form as follows:

Algorithm PGM

Step **1.** *solve* $p\,/\,NP$ systems with $A_s^{(k)}$ using Algorithm SM;
communications one_to_one: 1 vector of size n;
compute $\widetilde{\mathbf{f}}^{(2)}$;

Step **2.** *solve* incompletely $A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}$ using Algorithm PSRHS (for PGMS) or Algorithm PFSV (for PGMF);

Step **3.** *communications* one_to_one: 1 vector of size n;
compute $\widetilde{\mathbf{f}}^{(1)}$;
solve $p\,/\,NP$ systems with $A_s^{(k)}$ using Algorithm SM;

End {PGM}.

The first and the last steps are solution of p/NP systems with Algorithm SM, i.e. all the processors perform equal amount of work. To compute the related right–hand sides one vector of size n has to be transferred to one of the neighbors. Step 2 is handled either by PFSV or by PSRHS, which were discussed above. Note that the number of stages in PGMS at which communications are needed is a constant, while in PFSV and PGMF it depends on both size of the problem and used processors.

4 Numerical Tests

We implemented the presented parallel solvers in C using MPI standard. The achieved performance on two coarse-grained parallel platforms, referred to as Grendel and Beowulf, is analyzed. Grendel is a Silicon Graphics Origin 2000 with 8 R10000 processors at 250MHz, 4MB L2 cache and 4GB of RAM. Beowulf is a Beowulf cluster of 16 Digital (Compaq) Personal Workstations 500au with one EV56 processor at 500MHz and 128MB RAM per node, connected via 100Mbps switched Ethernet. To illustrate the properties of the considered algorithms and the related code on both machines, we present here results obtained for the case of variable coefficients of the problem (1).

Test example: The coefficients are $a_1(x_1) = 1 + x_1^2,\ a_2(x_2) = e^{-x_2}$. The function $u(x_1, x_2) = (1-x_1)x_1x_2(1-x_2)$ is taken for the solution $u(x_1, x_2)$ and the right-hand side $f(x_1, x_2) = 2x_2(1-x_2)(3x_1^2 - x_1 + 1) + e^{-x_2}x_1(1-x_1)(3-2x_2)$ corresponds to the above data.

The related discrete problem has $N = n^2$ degrees of freedom. It is obtained by five-point difference scheme applied on uniform $n \times n$ mesh with mesh parameter $h = 1/(n+1)$, and $m = n$. We use the decomposition $n + 1 = p(k + 1)$ for various p and $k \in \mathbf{Z}$. All the algorithms are direct and the error of the solution is in fact the error of the FDM approximation plus round-off. We compare in Table 1 the

Table 1: Results for the Sequential Algorithms

Platform	n	FSV	GMF		GMS		$\|u - u_h\|_{l_2}$
			k=3	k=7	k=3	k=7	
Grendel	255	3.12	3.02	2.90	1.99	1.02	8.43e-8
	511	19.01	19.29	11.66	18.64	7.94	2.11e-8
	1023	87.62	87.49	73.61	117.79	86.03	5.27e-9
Beowulf	255	6.91	6.95	4.02	4.48	2.53	8.43e-8
	511	28.57	28.42	16.88	23.04	12.79	2.11e-8
	1023	121.33	120.67	74.45	133.99	72.65	5.27e-9

computation times in seconds for the sequential algorithms on both platforms. First column tells the machine, and the second one - the value of n. The third column presents the cpu-time for the FSV algorithm. Next four columns are grouped by two and give the times for PGMF and PGMS for two different values of k, $k = 3$ and $k = 7$. For larger k the standard marching algorithm for this example is not stable. How the stability of SM affects the error for GMF is shown in [5]. In the last column is the discrete l_2 norm of the point-wise error of the computed solution, which up to a round–off should be one and the same for all of the considered methods. All reported

times in seconds in this section are the best ones obtained from 3 executions of the code on a given machine. We observe that the times on Grendel are smaller in general than those on Beowulf. But for both machines the behavior of the studied solvers is similar. Times for $k = 7$ for both GMF and GMS are better than those for $k = 3$. For small sized problems ($n = 255, n = 511$), the algorithm GMS is the fastest one. But, as it was expected, for $n = 1023$ it is the slowest one. The times for FSV are almost the same as those for $k = 3$ of GMF.

In the next two Tables 2 and 3 are reported the measured times for the related

Table 2: Results for the Parallel Algorithms on Grendel, $k = 7$

		PFSV			PGMF			PGMS		
n	NP	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}
255	2	2.47	–	–	1.60	–	–	1.07	–	–
	4	1.42	1.74	0.87	0.89	1.80	0.90	0.63	1.70	0.85
	8	1.08	2.29	0.57	0.65	2.46	0.62	0.37	2.89	0.72
511	2	16.86	–	–	10.52	–	–	7.59	–	–
	4	8.85	1.91	0.96	5.73	1.84	0.92	3.95	1.92	0.96
	8	4.98	3.39	0.85	3.28	3.21	0.80	2.24	3.39	0.85
1023	2	79.59	–	–	49.53	–	–	50.96	–	–
	4	41.63	1.91	0.96	26.13	1.90	0.95	24.00	2.12	1.06
	8	29.38	2.71	0.68	16.19	3.06	0.77	12.49	4.08	1.02

Table 3: Results for the Parallel Algorithms on Beowulf, $k = 7$

		PFSV			PGMF			PGMS		
n	NP	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}	T_{NP}	S_{NP}	E_{NP}
255	2	8.94	–	–	6.31	–	–	2.29	–	–
	4	3.90	2.29	1.15	2.72	2.32	1.16	1.74	1.32	0.66
	8	2.31	3.87	0.97	1.55	4.07	1.02	1.11	2.06	0.52
511	2	47.52	–	–	36.02	–	–	13.28	–	–
	4	23.40	2.03	1.02	17.83	2.02	1.01	8.51	1.56	0.78
	8	15.44	3.07	0.77	13.35	2.70	0.67	8.14	1.63	0.41
1023	2	271.13	–	–	227.04	–	–	89.46	–	–
	4	165.57	1.64	0.82	148.74	1.53	0.77	50.45	1.77	0.89
	8	126.88	2.14	0.53	125.18	1.81	0.45	41.07	2.18	0.55

parallel algorithms and the achieved speed-up and efficiency on Grendel and Beowulf respectively. The format of both tables is the following: first column characterizes the size of the problem – n is given where the number of unknowns is $N = n^2$. The second column shows the number NP of the used processors. The rest of the columns are in groups of three – one for each algorithm. In each group first column stands for the measured cpu-time. The case $k = 7$ is presented for the variants of the parallel GM. The second column in each group is for the speed-up $S_{NP} = \frac{T_2}{T_{NP}}$ and the third one is for the efficiency $E_{NP} = \frac{2T_2}{NPT_{NP}}$. The theoretical upper bounds for the speed-up and the efficiency for these cases are $S_{NP} \leq \frac{NP}{2}$ and $E_{NP} \leq 1$.

Let us first see what happens on Grendel (Table 2). The tendency for each of the algorithms is that S_{NP} and E_{NP} increase for larger size of the problem, and $E_4 > E_8$. There is an additional penalty in the speed-up (and the efficiency) for $NP = 8$ in the case $n = 1023$ because Grendel has only 8 processors. The algorithm with best speed-up for $NP = 4$ is changed when the value of n is varied. For the case $NP = 8$ the "winner" is always PGMS. The cpu-times for PGMS are the smallest for all values of n and NP.

Let us now look at the results on Beowulf (Table 3). Again cpu-times of PGMS are the smallest in all cases, but behavior of the speed-up and the efficiency is completely different. For algorithms PFSV and PGMF the efficiency decreases for larger values of n, while for PGMS it increases. The "winner" is again changed, but now PGMS is the looser for $n = 255$, 511 and $NP = 4$, 8. For large sized problems, i.e. $n = 1023$, PGMS has the best efficiency for both $NP = 4$, 8, although it is not close to the theoretical upper bound.

In the last Table 4 are compared times for PGMF and PGMS on Grendel for

Table 4: Comparison of PGMF and PGMS on Grendel, $k = 7$, T_{s2} - time for Step 2; T_r - time for Step 1 + Step3; T_i^c -time for communications (i=s2, r)

		PGMF					PGMS				
n	NP	T_{NP}	T_r	T_{s2}	T_r^c	T_{s2}^c	T_{NP}	T_r	T_{s2}	T_r^c	T_{s2}^c
	2	10.52	4.10	6.42	0.04	0.11	7.59	4.31	3.28	0.02	0.11
511	4	5.73	2.15	3.58	0.05	0.29	3.95	2.16	1.79	0.05	0.20
	8	3.28	1.16	2.12	0.10	0.40	2.24	1.11	1.13	0.05	0.30
	2	50.85	19.85	31.00	0.89	0.51	50.96	19.24	31.72	0.08	0.40
1023	4	26.69	10.22	16.47	0.91	0.73	24.00	9.31	14.69	0.11	0.77
	8	16.19	6.50	9.69	1.93	1.42	12.49	4.76	7.73	0.11	1.34

$n = 511$ and $n = 1023$. The second column tells the number of processors. Rest of the columns form two groups – one for each algorithm. The time for the whole algorithm is presented in the first one. The next two columns in each group show the time T_r for Step 1 and 3 and T_{s2} for Step 2 respectively. The last two columns are for the communication time again for Step 1 and 3 (T_r^c) and for Step 2 (T_{s2}^c). The time T_r is almost one and the same for both versions of GM and for the smaller size of the problem, Step 2 takes less time. Communication time is not very big on Grendel and hence it does not affect the speed-up as much as the total number of its processors.

Unfortunately, there were some problems on Beowulf and we were not able to fill similar table for it. Nevertheless, the presented results are enough to conclude that sometimes the slowest of our sequential solvers has the best parallel performance.

5 Concluding Remarks

The parallel performance of three direct separable solvers was compared. The behavior of the speed-up and efficiency of PFSV and PGMF was completely different on the different platforms we considered. For PGMS it was one and the same, although the values on Beowulf were smaller. The prediction that on some machines the slowest sequential solver will have the best parallel performance was confirmed.

Plans for future work on this topic include: a) MPI/OpenMP modification of the code (PFSV and both PGM); b)more experiments on (coarse- and fine-grained) shared memory, distributed memory and heterogeneous systems; c) generalizations of the considered solvers to 3D case; and d) development of efficient preconditioners (sequential and parallel) on the base of FSV and both GM.

Acknowledgments

The author is grateful to Dr. P. Vassilevski, Professor R. Lazarov, and Professor J. Pasciak who made this scientific visit possible and successful. The author is in debth to the Numerical Analysis group at TAMU, especially to T. Kolev and V. Dobrev, for the useful discussions concerning the solvers and related software. The numerical experiments were carried out on Grendel and Beowulf, located respectively in the Department of Mathematics and in the Institute for Scientific Computing, Texas A & M University.

References

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

[2] Bank, R., *Marching Algorithms for Elliptic Boundary Value Problems. II: The Variable Coefficient Case,* SIAM J. Numer. Anal. **14** (1977) 950–970.

[3] Bank, R., Rose, D., *Marching Algorithms for Elliptic Boundary Value Problems. I: The Constant Coefficient Case,* SIAM J. Numer. Anal. **14** (1977) 792–829.

[4] Bencheva, G., *Comparative performance analysis of 2D separable elliptic solvers,* Proceedings of the XIII-th summer school "Software and algorithms of numerical mathematics", Nectiny, Czech Republic, (1999), 11–27.

[5] Bencheva, G., *Comparative Analysis of Marching Algorithms for Separable Elliptic Problems,* Numerical Analysis and its Applications, (L. Vulkov, J. Wasniewski, P. Yalamov, eds.), Springer LNCS, **1988**, (2001), 76–83.

[6] Bencheva, G., *MPI Parallel Implementation of a Fast Separable Solver,* Large-Scale Scientific Computing, (S. Margenov, J. Wasniewski, P. Yalamov, eds.), Springer LNCS, **2179**, (2001), 454–461.

[7] Bencheva, G., *MPI Parallel Implementation of a Generalized Marching Algorithm,* in preparation.

[8] Kuznetsov, Y., *Block relaxation methods in subspaces, their optimization and application,* Sov. J. Numer. Anal. Math. Model. **4** (1989) 433–452.

[9] Kuznetsov, Y., Matsokin, A.M., *On partial solution of systems of linear algebraic equations,* Vychislitel'nye Motody Lineinoi Algebry (Ed. G.I. Marchuk). Vychisl. Tsentr Sib. Otdel. Akad. Nauk SSSR, Novosibirsk, (1978), pp. 62–89. In Russian, English translation in: Sov. J. Numer. Anal. Math. Modelling **4** (1989) 453–468.

[10] Sv. Petrova, *Parallel implementation of fast elliptic solver*, Parallel Computing **23** (1997) 1113–1128.

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

[12] Rossi, T., Toivanen, J., *A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension,* SIAM J. Sci. Comput., **20** (1999) No. 5, 1778–1796.

[13] Vassilevski, P.S., *An Optimal Stabilization of Marching Algorithm,* Compt. rend. de l'Acad. bulg. Sci. **41** (1988) No 7, 29–32.

[14] Vassilevski, P.S., *Fast Algorithm for Solving a Linear Algebraic Problem with Separable Variables,* Compt. rend. de l'Acad. bulg. Sci. **37** (1984) No 3, 305–308.