Parallel Implementation of a Generalized Marching Algorithm

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

August 20, 2004

Abstract

New parallel implementations of the generalized marching algorithm and its modification are theoretically studied. A decomposition of the computational domain into a number of strips corresponding to the number of processors is used in both cases. Estimates for the parallel times are derived. They are analyzed with respect to the problem size, the number of processors and the machine dependent parameters. The comparison of the properties of the considered algorithms shows that not always the best sequential solver has the best parallel performance.

1 Introduction

In this paper we pose the question: "Under what circumstances does an "expensive" direct elliptic solver lead to a better parallel algorithm than a "cheaper" one?" Our attention is focused on separable elliptic problems and more specifically on a parallel implementation of the generalized marching algorithm and its modification. More precisely, a separable second order elliptic equation with nonconstant coefficients is considered:

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

It is discretized on rectangular $n \times m$ grid by central finite differences or by piece-wise linear finite elements on right–angled triangles. Using the identity $n \times n$ matrix I_n, the

*Supported in part by the Ministry of Education and Science of Bulgaria under Grant #MY-I-901/99, by the Center of Excellence BIS-21 Grant ICA1-2000-70016 and by the USA National Science Foundation under Grant DMS 9973328

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 discrete system takes the following matrix form:

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

where $\mathbf{x}_j, \mathbf{f}_j \in \mathbf{R}^n$, $j = 1, \ldots, m$ are column vectors and $\mathbf{x}^T = \left(\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_m^T\right)$, $\mathbf{f}^T = \left(\mathbf{f}_1^T, \mathbf{f}_2^T, \ldots, \mathbf{f}_m^T\right)$. Lexicographic ordering on horizontal lines for the vectors $\mathbf{x}$ and $\mathbf{f}$ is used. This in fact means that A is block tridiagonal matrix with tridiagonal blocks on the main diagonal and the discrete problem has $N = nm$ degrees of freedom.

The generalized marching algorithm (GM) for direct solution of problems of type (2) is a stabilized version of the standard marching algorithm (SM), first proposed by R. Bank and D. Rose in [2, 3]. It is later reformulated by P. S. Vassilevski (in [12]) using fast algorithm for separation of variables (FASV) in combination with the so-called incomplete solution technique for problems with sparse right–hand sides (SRHS), proposed independently by Banegas [1], Proskurowski [10] and Kuznetsov [7]. More theoretical aspects of the problems with sparsity are investigated by Kuznetsov in [6]. Algorithm FASV itself (proposed in [13]) is another direct solver for system (2) also based on SRHS.

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

To obtain a good parallel implementation of the GMF algorithm, i.e. GM as it was proposed in [12], we have to use parallel implementation of FASV (PFASV). But on the other hand, PFASV requires parallelization of SRHS (see, e.g. [4, 11]) to be applied at some of its steps depending on the problem size and the number of processors. Hence, as it was shown in [4], c_1 for PFASV will depend not only on the number of processors but also on the size of the problem and PGMF will be influenced by this fact. What will happen if parallelization of SRHS is used instead of PFASV? In [5] it is shown experimentally that on some parallel systems the resulting GMS algorithm has better parallel performance than PGMF. This paper is focused on theoretical analysis and comparison of the parallel properties of GMF and its modification GMS.

The exposition is organized as follows. The algorithms SRHS, SM, GMF and GMS are briefly outlined in Section 2. Section 3 is devoted to the parallel implementation of GMF and GMS and to some theoretical analysis of their properties. Estimates for the execution times are derived and their behaviour is compared with respect to the machine dependent parameters t_s and t_w. Some concluding remarks are drawn at the end.

2 Generalized Marching Algorithm

We start the presentation in this section with a brief description of the technique for incomplete solution of systems of the form (2) with a sparse right–hand side (SRHS). That technique has independently been proposed by Banegas [1], Proskurowski [10], and Kuznetsov [7]. We next present the essence of the standard (SM) and generalized (GM) marching algorithms. The method GM is a stabilized version of the SM, first developed by Bank and Rose [2, 3] and later reformulated by Vassilevski [12] using SRHS and another fast separable elliptic solver called fast algorithm for separation of variables (FASV). We first present the latter version of GM and denote it as GMF. At the end of the section we describe the introduced in [5] modification of GM (referred as GMS) based only on SRHS and we give a short motivation for this approach.

2.1 Incomplete Solution Technique

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 that only r ($r \ll m$) block components of the solution are needed. Let for definiteness $\mathbf{f}_j = 0$ for $j \neq j_1, j_2, \ldots, j_d$. Then each vector $\mathbf{f}'_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,m})^T$, $i = 1, \ldots, n$ of the reordered right–hand side has only d nonzero scalar entries f_{i,j_s}, $s = 1, \ldots, d$. 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 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 $\{\mathbf{q}_{k,j}\}$, $j \in \{j_1, \ldots, j_d\} \cup \{j'_1, \ldots, j'_r\}$ 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 equations:

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

Step **2.** *solve* m $n \times n$ tridiagonal systems of linear equations for η_k ($\beta_k = (\beta_{1,k}, \ldots, \beta_{n,k})^T$ are given):

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

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

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

End{SRHS}.

Steps 1 and 3 require $2\,d\,m\,n$ and $2\,r\,m\,n$ arithmetic operations (ar. ops.) respectively, and $m(5\,n - 4)$ ar. ops. are needed for solution of the systems at Step 2.

The computational complexity of *Algorithm SRHS* is given in:

Lemma 2.1 *The Algorithm SRHS requires* $m[2(r + d)n + (5n - 4)]$ *ar. ops. in the solution part,* $m(4n-3)$ *ar. ops. to factor the tridiagonal matrices* $\lambda_k I_n + T$, $k = 1, \ldots, m$ *in* $LD^{-1}U$ *form, and* $\mathcal{O}(\tilde{d}\, m^2) + 9m^2$ *ar. ops. to compute all the eigenvalues and* $\tilde{d}$ *components of all the eigenvectors of the matrix* B.

2.2 Marching Algorithm (SM)

A simple rearrangement of system (2) is at the heart of the standard marching algorithm. The first block equation is placed at the bottom and the reordered system is rewritten in the following two-by-two block form:

$$\begin{pmatrix} U & G \\ C & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{x}_m \end{pmatrix} = \begin{pmatrix} \mathbf{f}' \\ \mathbf{f}_1 \end{pmatrix}. \tag{3}$$

Here, matrix U is upper triangular and the block matrices G and C, and vectors $\mathbf{x}'$ and $\mathbf{f}'$ read as follows

$$G = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ b_{m-1,m} I_n \\ T + b_{m,m} I_n \end{pmatrix}, \qquad \begin{aligned} C &= (T + b_{1,1} I_n \, , \, b_{1,2} I_n \, , \, 0 \, , \, \ldots \, , \, 0) \, , \\ \mathbf{x}'^T &= (\mathbf{x}_1^T \, , \, \mathbf{x}_2^T \, , \, \ldots \, , \, \mathbf{x}_{m-1}^T) \, , \\ \mathbf{f}'^T &= (\mathbf{f}_2^T \, , \, \mathbf{f}_3^T \, , \, \ldots \, , \, \mathbf{f}_m^T) \, . \end{aligned}$$

Using block-Gaussian elimination the problem (3) is reduced to the solution of two systems with the upper triangular matrix U and one system with the Schur complement $S = -C\,U^{-1}\,G$.

The steps of the standard marching algorithm are summarized below.

Algorithm SM

Step **1.** *solve* the system $U\mathbf{y}_1 = \mathbf{f}'$ using standard backward recurrence;
compute the right–hand side $\widetilde{\mathbf{f}}_1 = \mathbf{f}_1 - C\mathbf{y}_1$;

Step **2.** *solve* incompletely $A\,\widetilde{\mathbf{x}} = (\widetilde{\mathbf{f}}_1^T, \mathbf{0}^T, \ldots, \mathbf{0}^T)^T$ only for $\widetilde{\mathbf{x}}_m$
using *Algorithm SRHS* (with $d = 1, j_1 = 1$ and $r = 1, j_1' = m$)
to compute the solution $\mathbf{x}_m = \widetilde{\mathbf{x}}_m$ of the Schur complement system
$S\,\mathbf{x}_m = \widetilde{\mathbf{f}}_1$;

Step **3.** *determine* $\widehat{\mathbf{f}}' = -G\mathbf{x}_m + \mathbf{f}'$;
solve $U\mathbf{x}' = \widehat{\mathbf{f}}'$ using standard backward recurrence.

End{*SM*}.

Step 1 and Step 3 require $2((m-2)(11n-4)+n)$ ar. ops. for the systems with the upper triangular matrix U and $17n-8$ ar. ops. to compute the related right–hand sides. Step 2, according to Lemma 2.1, requires $m(9n-4)$ ar. ops. in the solution part. Hence, the next theorem for the computational complexity of *Algorithm SM* holds.

Theorem 2.1 *The marching algorithm in combination with the separation of variables technique for the incomplete solution of the reduced system requires an optimal cost* $\mathcal{N}_{SM} \approx 31\,n\,m$ *of ar. ops. for solving problems with separable variables (2).*

2.3 Generalized Marching Algorithm (GM)

The main disadvantage of SM as shown in [2, 3] is that it is unstable for large m. More precisely, the backward recurrence is unstable when it is applied to systems with the

upper triangular matrix U of large block order m. This makes SM of practical interest only if the length of this recurrence is small, i.e. for $m \ll n$. For the case $m \approx n$ one may use the generalized marching algorithm, described below.

For ease of presentation, let $m+1 = p(k+1)$ for some integers p and k. Similar to the standard marching algorithm, the system (2) is first reordered and rewritten into two-by-two block form

$$\begin{pmatrix} \widetilde{A}_{1,1} & \widetilde{A}_{1,2} \\ \widetilde{A}_{2,1} & \widetilde{A}_{2,2} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^{(1)} \\ \mathbf{f}^{(2)} \end{pmatrix}.$$

Then applying again block-Gaussian elimination, it is reduced to solution of two systems with the block $\widetilde{A}_{1,1}$ and one system with the Schur complement $S = \widetilde{A}_{2,2} - \widetilde{A}_{2,1}\,\widetilde{A}_{1,1}^{-1}\,\widetilde{A}_{1,2}$. Now, $\widetilde{A} = (\widetilde{A}_{i,j})_{i,j=1}^{2}$ is a symmetric block odd–even reordering of A. It is induced by simultaneous reordering of the unknown vector $\mathbf{x}^T = (\mathbf{x}^{(1)^T}, \mathbf{x}^{(2)^T})$ and the right–hand side $\mathbf{f}^T = (\mathbf{f}^{(1)^T}, \mathbf{f}^{(2)^T})$ by the rule: all rows of $\mathbf{x}$ (respectively of $\mathbf{f}$) of multiplicity $k+1$ form its second block component $\mathbf{x}^{(2)}$. The block $\mathbf{x}^{(2)}$ is a separator – it partitions the $n \times m$ grid into p strips with k grid lines. Hence the blocks on the diagonal of $\widetilde{A}$ are block–diagonal. More specifically, they are defined by:

$$\begin{aligned} \widetilde{A}_{1,1} &= \mathit{blockdiag}(A_s^{(k)})_{s=1}^{p},\ A_s^{(k)} = I_k \otimes T + B_s^{(k)} \otimes I_n, \\ B_s^{(k)} &= \mathit{tridiag}(b_{k_s+i,k_s+i-1}\,, b_{k_s+i,k_s+i}\,, b_{k_s+i,k_s+i+1})_{i=1}^{k}\,, \\ \widetilde{A}_{2,2} &= \mathit{blockdiag}(T + b_{k_{s+1},k_{s+1}}\, I_n),\ k_s = (s-1)(k+1), s = 1, \ldots, p\,. \end{aligned}$$

The components $\mathbf{x}^{(i)}, \mathbf{f}^{(i)}, i = 1, 2$ of the solution and the right–hand side are grouped as follows ($k_s = (s-1)(k+1)$, $s = 1, \ldots, p$):

$$\mathbf{x}^{(1)} = \begin{pmatrix} \mathbf{x}_1^{(1)} \\ \vdots \\ \mathbf{x}_p^{(1)} \end{pmatrix},\ \mathbf{f}^{(1)} = \begin{pmatrix} \mathbf{f}_1^{(1)} \\ \vdots \\ \mathbf{f}_p^{(1)} \end{pmatrix},\ \mathbf{x}_s^{(1)} = \begin{pmatrix} \mathbf{x}_{k_s+1} \\ \vdots \\ \mathbf{x}_{k_s+k} \end{pmatrix},\ \mathbf{f}_s^{(1)} = \begin{pmatrix} \mathbf{f}_{k_s+1} \\ \vdots \\ \mathbf{f}_{k_s+k} \end{pmatrix},$$

$$\begin{aligned} \mathbf{x}^{(2)^T} &= (\mathbf{x}_{k+1}^T, \ldots, \mathbf{x}_{s(k+1)}^T, \ldots, \mathbf{x}_{(p-1)(k+1)}^T), \\ \mathbf{f}^{(2)^T} &= (\mathbf{f}_{k+1}^T, \ldots, \mathbf{f}_{s(k+1)}^T, \ldots, \mathbf{f}_{(p-1)(k+1)}^T). \end{aligned}$$

The subproblems with $A_s^{(k)}$ are independent of each other and the systems with $\widetilde{A}_{1,1}$ are solved by applying p times the *Algorithm SM*. To ensure the stability of SM at this step, the length $k-1 = \dfrac{m+1}{p} - 2$ of the backward recurrence is controlled by choosing sufficiently large p.

The system with the Schur complement is again equivalent to incomplete solution of a system with the original matrix and with a sparse right–hand side. Now, $p-1$ components of the solution are needed and the right–hand side has $p-1$ nonzero blocks.

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 the right–hand side $\widetilde{\mathbf{f}}^{(2)} = \mathbf{f}^{(2)} - \widetilde{A}_{2,1}\,\mathbf{y}^{(1)}$;

Step **2.** *solve* incompletely $A\widehat{\mathbf{x}} = \widehat{\mathbf{f}}$, where $\widehat{\mathbf{f}}_i = \begin{cases} \widetilde{\mathbf{f}}_s^{(2)}, & i = s(k+1) \\ 0, & i \neq s(k+1) \end{cases}$,
seeking only $\widehat{\mathbf{x}}_{s(k+1)} = \mathbf{x}_{s(k+1)}$, $s = 1, \ldots, p-1$
to compute the solution of the system $S\,\mathbf{x}^{(2)} = \widetilde{\mathbf{f}}^{(2)}$;

Step **3.** *compute* the right–hand side $\widetilde{\mathbf{f}}^{(1)} = \mathbf{f}^{(1)} - \widetilde{A}_{1,2}\,\mathbf{x}^{(2)}$;
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 1 and Step 3 are in fact solution of $2p$ systems of block order k using *Algorithm SM*. They require $2p(31kn - 25n - 12k + 8)$ ar. ops. The related right–hand sides are updated with $8n(p-1)$ ar. ops. These two steps require approximately $62pkn \approx 62mn$ ar. ops. The total cost of GM algorithm depends on how Step 2 is handled.

The variant of GM proposed in [12] and referred here as GMF uses $l_p = \log p$ steps of the algorithm FASV (developed in [13]). In this case $24nm(\log p - 1) - 9nm$ ar. ops. are needed for Step 2. The total cost of GMF is summarized in

Theorem 2.2 *The generalized marching algorithm in combination with the fast algorithm for separation of variables and the incomplete solution technique takes* $\mathcal{N}_{GMF} \approx 62\,mn + 24\,nm(\log p - 1) - 9\,nm$ *ar. ops. to solve systems of the form (2).*

Similar to the approach in SM, another possible way to solve the system with the Schur complement is to use *Algorithm SRHS* with $d = r = p - 1$. The resulting modification of GM is denoted for brevity with GMS. The estimates in Lemma 2.1 and Theorem 2.1 lead to the next Theorem for the computational complexity of algorithm GMS.

Theorem 2.3 *The modification GMS of the generalized marching algorithm based on the incomplete solution technique requires* $\mathcal{N}_{GMS} \approx 62\,mn + 4\,p\,nm + nm$ *ar. ops.*

For large values of $\log p$ ($\log p \geq 5$) GMF is faster than GMS. Then why do we introduce GMS? Let us first recall that Algorithm FASV is in some sense recursive (for more details see, e. g. [9]). It consists of forward and backward recurrence, each with $l = \log m$ steps. At each of the steps $s = 1, \ldots, l$, $q = 2^{l-s}$ systems of order $\dfrac{m}{q}$ are solved incompletely using SRHS with $r = 3$, $d = 1$ at the forward recurrence and with $r = 1$, $d = 2$ at the backward recurrence. Our goal in this study is to obtain a scalable parallel implementation of the generalized marching algorithm. When we start from GMF we have to use parallel implementation of FASV (PFASV). But on the other hand, the variable size of the subsystems solved incompletely in FASV lead to some difficulties in its parallelization. PFASV requires parallelization of SRHS (see, e.g. [4, 11]) to be applied at some of its steps depending on the problem size and the number

of used processors. If the starting sequential solver is GMS we need parallel SRHS only once. What we gain and what we lose with this modification will be discussed in the next Section.

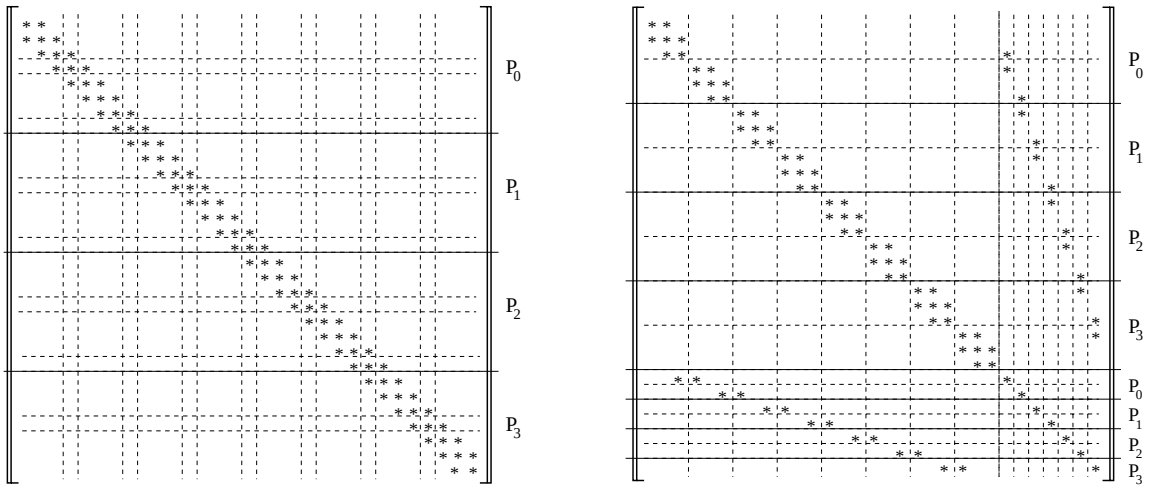
3 Parallel Implementation

Parallel implementations of the GMS (PGMS) and GMF (PGMF) algorithms are proposed and experimentally compared in [5]. Here they are described in some more details and their properties are theoretically studied. First part of this section deals with the questions "How to partition the data and computations?", "What kind of communications are induced by such division?". Both PGMS and PGMF, as well as parallel SRHS (PSRHS) as an important part of PGMS, are given in a compact form. The parallel times of PGMF and PGMS are evaluated in the second part, where the estimates for PFASV used in PGMF are taken from [4]. Their behaviour on the ring, 2D mesh and hypercube architectures is illustrated using the expressions for local and global communications given in the book of Kumar et. al. [8]. The advantages and disadvantages of PGMS versus PGMF are summarized in the last part. They are based on the presented theoretical analysis and our experience with the parallel FASV (see [4]) and the obtained in [5] numerical results for PFASV, PGMF and PGMS.

3.1 Data distribution, computations and communications

Let us first discuss how GMS could be partitioned into small tasks to be executed concurrently. At each of its stages there is a group of independent systems which may be solved in parallel. At Step 1 and Step 3 these are the p block equations with matrices $A_s^{(k)}$ of order kn (k blocks of order n). At Step 2 the m tridiagonal systems $(\lambda_i I_n + T)\eta_i = \beta_i$, $i = 1, \ldots, m$ of order n can be considered as separate tasks. In both cases the whole matrix T is needed. In addition, the related parts of the matrix B should be available for each of the tasks in the first group. In the second one these are some of the eigenvalues and parts of the eigenvectors of B. Taking these observations into account we divide data and computations in the following way. Let us have $N_p = 2^{n_p}$ processors enumerated with $P_0, P_1, \ldots, P_{N_p-1}$. We assume that $N_p \leq p$ and *Algorithm SM* need not to be implemented in parallel. The initial data, i. e. the matrix A and the right–hand side $\mathbf{f}$ (and the remainder of the vectors) are divided into N_p strips with approximately equal size – first $N_p - 1$ are of length $LSTRIP = 2^{l-n_p}$ and the last one is $LSTRIP - 1$ ($m + 1 = 2^l = p(k + 1)$). Two strips with common boundary are associated with processors with successive indices. Each of $P_i, i = 0, \ldots, N_p - 1$ contains the whole matrices T and B and the i-th block of $\mathbf{f}$. In such a way each processor can solve $\frac{p}{N_p}$ systems with $A_s^{(k)}$. It can also solve (at the preprocessing stage) the related eigenproblems and to store the data required for Step 2. The same partitioning is enough for PGMF. Note that the whole B is needed only to compute the eigenpairs at the preprocessing stage. After that it is enough each P_i to contain the i-th strip of B.

The links between equations in the original system are now in the blocks $\widetilde{A}_{1,2}$ and $\widetilde{A}_{2,1}$ of the reordered matrix $\widetilde{A} = \{\widetilde{A}_{i,j}\}_{i,j=1}^{2}$. To compute the related right–hand sides one have to perform *matrix* $\times$ *vector* multiplications with $\widetilde{A}_{1,2}$ and $\widetilde{A}_{2,1}$. How do these



A $\widetilde{A}$

Figure 1: Structure of A and $\widetilde{A}$ and distribution among processors when $p = 8, k = 3, m = p(k+1), N_p = 4$.

blocks look like? The structure of the matrices A and $\widetilde{A}$ and the distribution of their entries among the processors are presented in Figure 1 for the case $p = 8, k = 3, m = p(k+1), N_p = 4$. A nonzero block of order n is denoted by $*$. The right-hand side $\mathbf{f}$ and all the auxiliary vectors are distributed in a similar way. We know that $\widetilde{A}_{1,1}$ and $\widetilde{A}_{2,2}$ are block-diagonal – $\widetilde{A}_{1,1}$ has p nonzero blocks of order kn, while the blocks of $\widetilde{A}_{2,2}$ are $p-1$ of order n. Therefore, $\widetilde{A}_{1,2}$ and $\widetilde{A}_{2,1}$ have respectively $p \times (p-1)$ blocks of order $kn \times n$ and $(p-1) \times p$ blocks of order $n \times kn$. Each column of $\widetilde{A}_{1,2}$ has not more than 2 nonzero entries of order n, while in $\widetilde{A}_{2,1}$ the same holds for the rows. Then P_i $(i \neq 0)$ sends to P_{i-1} the first component of order n of $\mathbf{y}^{(1)}$ to ensure computation of $\widetilde{A}_{2,1}\mathbf{y}^{(1)}$, while to determine $\widetilde{A}_{1,2}\mathbf{x}^{(2)}$, P_i $(i \neq N_p - 1)$ sends to P_{i+1} the last block of $\mathbf{x}^{(2)}$. So at each of the Steps 1 and 3 for both PGMF and PGMS one vector of length n should be transferred between neighbours.

The difference between GMS and GMF is in the way Step 2 is handled. For PGMF is needed parallel implementation of FASV, while parallel SRHS is used in PGMS.

Now we are ready to present the compact form of PGM:

Algorithm PGM

Step **1.** *solve* $\dfrac{p}{N_p}$ 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 PFASV* (for PGMF);

Step **3.** *communications* one_to_one: 1 vector of size n;

compute $\widetilde{\mathbf{f}}^{(1)}$;
solve $\dfrac{p}{N_p}$ systems with $A_s^{(k)}$ using *Algorithm SM*;

End {*PGM*}.

Algorithm PFASV is proposed and studied in [4]. To design PSRHS we have to keep in mind the specific requirements for its input and output data. At the beginning of Step 2 of PGMS each processor will have a block of the right-hand side and the whole matrices T and B. At the end P_i should contain a group of $\dfrac{p}{N_p}$ (and $\dfrac{p}{N_p}-1$ for P_{N_p-1}) of the sought components $\mathbf{x}_{s(k+1)}$, $s=1,\ldots,p-1$. Note that the version of PSRHS presented below is slightly different from that proposed in [4] and used in parallel FASV, because the input and output data for PFASV and PGMS are not one and the same.

Algorithm PSRHS

Step **0.** *compute* all the eigenvalues $\{\lambda_i\}_{i=1}^m$ and the needed $\tilde{d} \le r+d$ entries $\{\mathbf{q}_{i,j}\}$, $j \in \{j_1,\ldots,j_d\} \cup \{j_1^{'},\ldots,j_r^{'}\}$ of all the eigenvectors $\{\mathbf{q}_i\}_{i=1}^m$ of the tridiagonal matrix B ; (preprocessing stage);

Step **1.** *determine* the Fourier coefficients $\beta_{i,j}$ by

computations: $\beta_{i,j}^{(Pr)} = \sum_{s=1}^{d(Pr)} q_{j_s,j} f_{i,j_s}$, $i=1,\ldots,n$, $j=1,\ldots,m$;

communications: all_to_all reduce_scatter for β_j;

Step **2.** *solve* $(\lambda_j I_n + T)\,\eta_j = \beta_j$, $j=1,\ldots,\dfrac{m}{N_p}$;

Step **3.** *recover* $\mathbf{x}_j$, $j=j_1',j_2',\ldots,j_r'$ by

computations: $\mathbf{x}_j^{(Pr)} = \sum_{i=1}^{\widetilde{m}} q_{j,i}\eta_i$, $j=j_1',\ldots,j_r', \widetilde{m}=\dfrac{m}{N_p}$;

communications: all_to_all reduce_scatter for $\mathbf{x}_j$

End {*PSRHS*}.

At the preprocessing stage each processor solves the eigenproblem and stores the data in the required way. More specifically, for Step 2 each processor needs a block of length $\dfrac{m}{N_p}$ of the eigenvalues $\lambda = \{\lambda_i\}_{i=1}^m$. For Step 1, i.e. to compute $\beta_{i,j}$, each of P_i should have $d(Pr) = \dfrac{d}{N_p}$ components of all the eigenvectors. For Step 3 P_i needs r entries of $\dfrac{m}{N_p}$ eigenvectors. Computations and communications at Step 1 are the same as if we have to perform *matrix* $\times$ *matrix* multiplication. The data for this multiplication is distributed as follows:

$$\underset{F_{n\times d}}{\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right)} \quad \underset{Q_{d\times m}}{\left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right)} \quad = \quad \underset{Beta_{n\times m}}{\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right)} .$$

Here left-hand matrix is divided in strips by columns, the right-hand one – by rows, and the product should be distributed among processors again in strips by columns. Each

processor computes the parts $\beta_{i,j}^{(Pr)}$ of $\beta_{i,j}$ and after that P_i, $i = 0, \ldots, N_p - 1$ collects $\frac{m}{N_p}$ of $\beta_j = \sum_{Pr} \beta_j^{(Pr)}$. The required global communications are denoted by all_to_all reduce_scatter and they are performed as follows. First all_to_one reduce is executed and a single processor contains all the vectors β_j. Next their parts are transferred to processors which will use them by one_to_all scatter. At the next Step 2 each processor solves $\frac{m}{N_p}$ independent tridiagonal systems without any communications. The third step is performed in a similar way as Step 1 with the only difference that the second matrix and the result are now column vectors:

$$\underbrace{\left(\begin{array}{c|c|c|c} P_0 & P_1 & P_2 & P_3 \\ & & & \end{array}\right)}_{Q_{r\times m}} \quad \underbrace{\left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right)}_{V_m} = \underbrace{\left(\begin{array}{c} P_0 \\ \hline P_1 \\ \hline P_2 \\ \hline P_3 \end{array}\right)}_{X_r}.$$

This time parts of $\mathbf{x}_j$ are determined and after that each processor obtains its components $\mathbf{x}_j = \sum_{Pr} \mathbf{x}_j^{(Pr)}$ with the help of all_to_all reduce_scatter.

3.2 Parallel times estimates

How to evaluate the execution time T_{N_p} for the considered solvers? It will depend not only on the particular algorithm, but also on the used parallel computer and its characteristics. We assume that we have a distributed memory machine with $N_p = 2^{n_p}$ processors. Let us recall that $p = 2^{l_p}$, $k + 1 = 2^{l_k}$, $m + 1 = 2^l$, $l = l_k + l_p$. For the machine dependent parameters we use the following notations: t_a is the average unit time to perform 1 ar. op. on one processor, t_s is the start-up time to initiate a communication, and t_w is the time necessary to send a word between two processors. We suppose also that computations and communications are not overlapped and hence the total time per processor is $T_{N_p} = T_a + T_{com}$. We assume that there is no vectorization and the execution of M ar. ops. on a single processor takes time $T_a = M * \mathbf{t}_a$. The communication time is presented as $T_{com} = c_1 * \mathbf{t}_s + c_2 * \mathbf{t}_w$, where the parameters c_1 and c_2 depend on the type and the amount of communications. The following assumptions are taken into account to derive c_1 and c_2 for our solvers. To send M words between two processors, i.e. to perform one_to_one communication we need time $T_{oo} = t_s + l.M * t_w$, where l is the distance between them and when they are neighbours it is $T_{loc} = t_s + M * t_w$. We suppose that processors with successive indices are physically neighbours, i.e. the communications between them are local. Estimates for the time of the global communications one_to_all broadcast, all_to_one reduce and one_to_all scatter are also needed. The time to broadcast a package of M words to a group of N_p processors is denoted with $b(N_p, M)$. The scheme for execution of one_to_all broadcast and all_to_one reduce is one and the same but the operations are performed in a reverse order and hence they take equal amount of time. When a processor send N_p packages of M words

(one per receiver) using one_to_all scatter, the spent time is $s(N_p, M)$. Upper bounds for $T_{oo} = oo(N_p, M)$, $b(N_p, M)$ and $s(N_p, M)$ on ring, 2D mesh and hypercube architectures are given in Table 1 (details about how they are obtained could be found in [8]).

Table 1: Estimates for basic local and global communications

	Ring	2D mesh	hypercube
$oo(N_p, M)$	$t_s + t_w.M.\lfloor N_p/2 \rfloor$	$t_s + 2t_w.M.\lfloor \sqrt{N_p}/2 \rfloor$	$t_s + t_w.M.\log N_p$
$b(N_p, M)$	$(t_s + t_w.M)\lceil N_p/2 \rceil$	$2(t_s + t_w.M)\lceil \sqrt{N_p}/2 \rceil$	$(t_s + t_w.M)\log N_p$
$s(N_p, M)$	$(t_s + t_w.M)(N_p - 1)$	$2t_s(\sqrt{N_p} - 1) +$ $t_w.M(N_p - 1)$	$t_s \log N_p +$ $t_w.M.(N_p - 1)$

We are ready to derive theoretical estimates for the parallel times of PGMS and PGMF. First we focus our attention on PSRHS as an important part of PGMS.

Theorem 3.1 *The parallel implementation PSRHS of Algorithm SRHS for solution of the problem (2) with a sparse right-hand side with d nonzero entries, when only r block components of the solution are needed using N_p processors is executed for time $T^{psg}(N_p, m, r, d)$, where*

$$\begin{aligned} T^{psg}(N_p, m, r, d) &= T_a^{psg}(N_p, m, r, d) + T_{com}^{psg}(N_p, m, r, d)\,, \\ T_a^{psg}(N_p, m, r, d) &= \left(2\frac{mn}{N_p}(r+d) + \frac{m}{N_p}(5n-4)\right) * t_a\,, \\ T_{com}^{psg}(N_p, m, r, d) &= b(N_p, m.n) + s(N_p, \frac{m}{N_p}n) + b(N_p, r.n) + s(N_p, \frac{r}{N_p}n)\,. \end{aligned}$$

Proof. Computations: At Step 1 are performed $2\frac{d}{N_p}mn$ ar. ops. to determine the related Fourier coefficients. At Step 2 each processor solves $\frac{m}{N_p}$ linear algebraic systems with $\frac{m}{N_p}(5n-4)$ ar. ops. The solution is found at Step 3 by $2r\frac{m}{N_p}n$ ar. ops. Total computation time for PSRHS is $T_a^{psg}(N_p, m, r, d) = \left(2\frac{mn}{N_p}(r+d) + \frac{m}{N_p}(5n-4)\right) * t_a = \frac{\mathcal{N}_{SRHS}}{N_p} * t_a$. *Communications:* At each of the Steps 1 and 3 a global communication denoted as all_to_all reduce_scatter is performed. At Step 1 it is used to send parts of the Fourier coefficients – all_to_one reduce for m vectors, each of length n is followed by one_to_all scatter for packages containing $\frac{m}{N_p}$ vectors of length n which takes time $b(N_p, m.n) + s(N_p, \frac{m}{N_p}n)$. Parts of the solution are transferred at Step 3 in a similar way for time $b(N_p, r.n) + s(N_p, \frac{r}{N_p}n)$. ∎

Only l_p steps of algorithm PFASV are needed in PGMF. Taking into account this fact we modify the estimates for PFASV obtained in [4]. We get $T_{inc}^{pfasv}(N_p,\, m) = T_{a,\,inc}^{pfasv}(N_p,\, m) + T_{com,\,inc}^{pfasv}(N_p,\, m)$, where $\widehat{m} = \dfrac{m+1}{2^{n_p-j}} - 1$ and

$$\begin{aligned}
T_{a,\,inc}^{pfasv}(N_p,\, m) &= \Big(24n\frac{m}{N_p}(l_p-1) + 9n\frac{m-2p}{N_p}\Big) * t_a\,,\\
T_{com,\,inc}^{pfasv}(N_p,\, m) &\le (l_p - n_p)(t_s + n * t_w) + T_{com}^{psf}(2^{n_p},\, m,\, 1,\, 1) + n_p\, oo(2^{n_p},\, n)\\
&+ \sum_{j=1}^{n_p-1} (T_{com}^{psf}(2^j,\, \widehat{m},\, 3,\, 1) + T_{com}^{psf}(2^j,\, \widehat{m},\, 1,\, 2) + (2+j)\, oo(2^j,\, n)).
\end{aligned}$$

Note that the parallel implementation of SRHS used in PFASV is slightly different from that proposed here. Its execution time is $T^{psf}(N_p,\, m,\, r,\, d) = T_a^{psf}(N_p,\, m,\, r,\, d) + T_{com}^{psf}(N_p,\, m,\, r,\, d)$, where

$$\begin{aligned}
T_a^{psf}(N_p,\, m,\, r,\, d) &= T_a^{psg}(N_p,\, m,\, r,\, d) = \left(2\frac{mn}{N_p}(r+d) + \frac{m}{N_p}(5n-4)\right) * t_a\,,\\
T_{com}^{psf}(N_p,\, m,\, r,\, d) &= b(N_p,\, d.n) + b(N_p,\, r.n)\,.
\end{aligned}$$

For ring, 2D mesh and hypercube (denoted with upper indices r, m, h) the communication time for both versions of PSRHS is bounded respectively by:

$$\begin{aligned}
T_{com}^{psf,\,r}(N_p,\, m,\, r,\, d) &= 2\left\lceil\frac{N_p}{2}\right\rceil * t_s + \left\lceil\frac{N_p}{2}\right\rceil c_1 * t_w\,,\\
T_{com}^{psg,\,r}(N_p,\, m,\, r,\, d) &= 2\Big(N_p - 1 + \left\lceil\frac{N_p}{2}\right\rceil\Big) * t_s + \Big(c_2 + \left\lceil\frac{N_p}{2}\right\rceil c_3\Big) * t_w\,,\\
T_{com}^{psf,\,m}(N_p,\, m,\, r,\, d) &= 4\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil * t_s + 2\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil c_1 * t_w,\\
T_{com}^{psg,\,m}(N_p,\, m,\, r,\, d) &= 4\Big(\sqrt{N_p} - 1 + \left\lceil\frac{\sqrt{N_p}}{2}\right\rceil\Big) * t_s + \Big(c_2 + 2\left\lceil\frac{\sqrt{N_p}}{2}\right\rceil c_3\Big) * t_w,\\
T_{com}^{psf,\,h}(N_p,\, m,\, r,\, d) &= 2\log N_p * t_s + c_1 \log N_p * t_w\,,\\
T_{com}^{psg,\,h}(N_p,\, m,\, r,\, d) &= 4\log N_p * t_s + (c_2 + c_3 \log N_p) * t_w\,,
\end{aligned}$$

where $c_1 = (d+r)n$, $c_2 = \dfrac{N_p-1}{N_p}(m+r)n$, $c_3 = (m+r)n$.

These observations are related to Step 2 of PGMF and PGMS. What happens at Step 1 and Step 3? At each of them $\dfrac{p}{N_p}$ systems of order k are solved using *Algorithm SM*, i. e. $\dfrac{p}{N_p}31kn$ ar. ops. are performed. To compute the related right-hand sides $4n\dfrac{p-1}{N_p}$ ar. ops. are needed. In total, Steps 1 and 3 of either PGMF or PGMS require

$\mathcal{N}_{1\&3}^{pgm} \approx 62\frac{pk}{N_p}n = 62\frac{m}{N_p}n$ ar. ops. To compute the right-hand sides at each of these steps also local communications have to be preformed – one vector of length n is transferred between processors with successive indices, which by assumption are neighbours. Thus the communication time here is $T_{com,1\&3}^{pgm} = 2(t_s + n * t_w)$ and the total time for Steps 1 and 3 is $T_{1\&3}^{pgm} = 62\frac{pk}{N_p}n * t_a + 2(t_s + n * t_w)$.

Hence for the execution times of PGMF and PGMS the following theorem holds.

Theorem 3.2 *The parallel implementation PGMF of the generalized marching algorithm in combination with the fast algorithm for separation of variables is executed for time* $T^{pgmf}(N_p,\, m) = T_a^{pgmf}(N_p,\, m) + T_{com}^{pgmf}(N_p,\, m)$. *The modification PGMS, using only the incomplete solution technique for problems with sparse right-hand sides requires time* $T^{pgms}(N_p,\, m) = T_a^{pgms}(N_p,\, m) + T_{com}^{pgms}(N_p,\, m)$. *The additives in these expressions read as follows (*$\widehat{m} = \dfrac{m+1}{2^{n_p-j}} - 1$*):*

$$
\begin{aligned}
T_a^{pgmf}(N_p,\, m) &= \Big(62\frac{pk}{N_p}n + 24n\frac{m}{N_p}(l_p - 1) + 9n\frac{m-2p}{N_p}\Big) * t_a\,, \\
T_a^{pgms}(N_p,\, m) &= \Big(62\frac{pk}{N_p}n + \frac{m}{N_p}n(4p+1)\Big) * t_a\,, \\
T_{com}^{pgmf}(N_p,\, m) &\le (l_p - n_p + 2)(t_s + n * t_w) + T_{com}^{psf}(2^{n_p},\, m,\, 1,\, 1) + n_p\, oo(2^{n_p},\, n) \\
&+ \sum_{j=1}^{n_p-1} (T_{com}^{psf}(2^j,\, \widehat{m},\, 3,\, 1) + T_{com}^{psf}(2^j,\, \widehat{m},\, 1,\, 2) + (2+j)\, oo(2^j,\, n)), \\
T_{com}^{pgms}(N_p,\, m) &\le 2(t_s + n * t_w) + T_{com}^{psg}(N_p,\, m, p-1, p-1)\,.
\end{aligned}
$$

Computations are distributed into equal parts among the processors. For ring, 2D mesh and hypercube, the communication times for PGMF are estimated with

$$
\begin{aligned}
T_{com}^{pgmf,r}(N_p,\, m) &\le \Big(l_p - 4 + 3.2^{n_p} + \frac{n_p(n_p+3)}{2}\Big) * t_s \\
&+ n(l_p - n_p - 6 + 2^{n_p-1}(2n_p+9)) * t_w\,, \\
T_{com}^{pgmf,m}(N_p,\, m) &\le \Big(l_p - 4 + \lfloor f_1(n_p)\rfloor + \frac{n_p(n_p+19)}{2}\Big) * t_s \\
&+ n(l_p + 13n_p - 8 + \lfloor f_2(n_p)\rfloor) * t_w\,, \\
T_{com}^{pgmf,h}(N_p,\, m) &\le (l_p + \frac{n_p(5n_p+3)}{2}) * t_s + n(l_p + 2 + \frac{n_p(n_p^2+15n_p-10)}{3}) * t_w,
\end{aligned}
$$

where $f_1(n_p) = -4(2+\sqrt{2}) + 2\sqrt{2}^{n_p}(3+2\sqrt{2})$ and $f_2(n_p) = -14 - 6\sqrt{2} + \sqrt{2}^{n_p}(7 + 6\sqrt{2} + (2+\sqrt{2})n_p)$. For PGMS these times are

$$
T_{com}^{pgms,r}(N_p,\, m) \le 2\Big(N_p + \Big\lceil \frac{N_p}{2} \Big\rceil\Big) * t_s + \Big(2n + g(c + \Big\lceil \frac{N_p}{2} \Big\rceil)\Big) * t_w\,,
$$

$$T_{com}^{pgms,m}(N_p,\, m) \leq \left(4\sqrt{N_p} - 2 + 4\left\lceil \frac{\sqrt{N_p}}{2} \right\rceil\right) * t_s + \left(2n + g(c + 2\left\lceil \frac{\sqrt{N_p}}{2} \right\rceil)\right) * t_w,$$

$$T_{com}^{pgms,h}(N_p,\, m) \leq (4\log N_p + 2) * t_s + (2n + g(c + \log N_p)) * t_w\,,$$

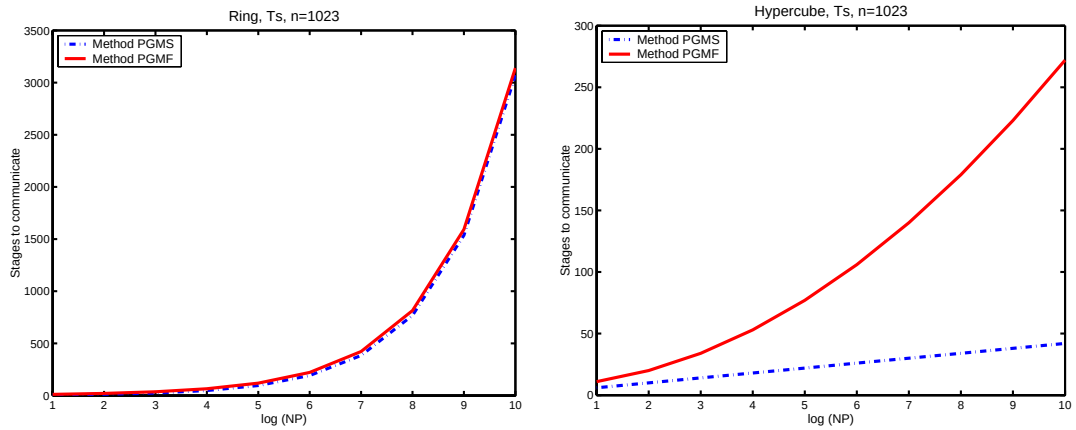
where $g = (m + p - 1)n$ and $c = \dfrac{N_p - 1}{N_p}$.

3.3 Parallel GMF versus parallel GMS

What are the advantages and the disadvantages of the studied solvers? The time for execution of each solver, and respectively the speed-up and the efficiency, depends on both calculations and type and amount of the required communications. For shared memory parallel computers, the time for computations is the leading term in $T^{Alg}(N_p) = T_a(N_p) + T_{com}(N_p)$. The computations are divided into equal parts among processors and $T_a(N_p) = 2T_a(2N_p)$ for both PGMF and PGMS. So the algorithm with smaller operation count (PGMF in our case) is better when the influence of $T_{com}(N_p)$ is negligible.

For distributed memory systems, the performance depends to a great extent on $T_{com}(N_p)$. We have obtained in 3.2 estimates of the form $T_{com}(N_p) \leq T_s * t_s + T_w * t_w$ for three parallel architectures – ring, 2D mesh and hypercube. The coefficients T_s and T_w represent respectively the number of stages at which communications are needed and the total amount of the transferred words. They depend in addition on the type of the architecture, since for the considered solvers we have both global and local communications.

The values of T_s as a function of $n_p\,(N_p = 2^{n_p})$ for $m = n = 1023$ for ring and hypercube are presented in Figure 2. Both graphics for ring almost coincide with a

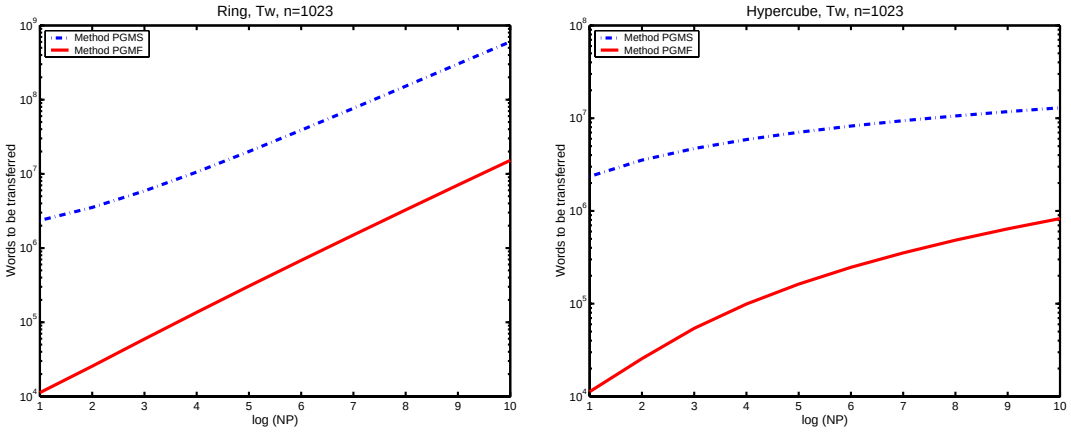


T_s^r T_s^h

Figure 2: Coefficients in front of t_s, for $k = 7\,,\ n = 1023$.

slight advantage of PGMS. The behaviour of T_s on 2D mesh is similar since the leading terms (as for ring) for both algorithms are equal. For hypercube, as shown in Figure 2,

T_s^{pgms} grows up very slowly compared to T_s^{pgmf}. The reason is that the leading term for PGMS is n_p, while for PGMF it is n_p^2.



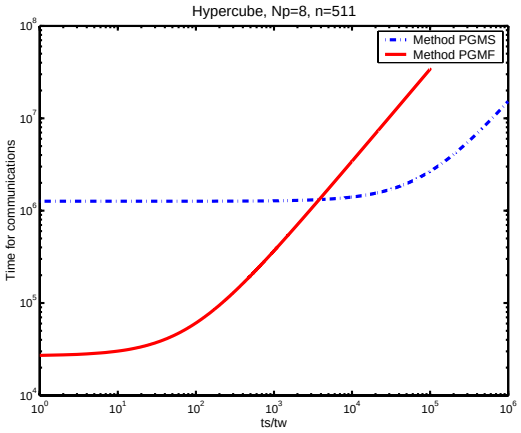
T_w^r T_w^h

Figure 3: Coefficients in front of t_w, for $k = 7$, $n = 1023$.

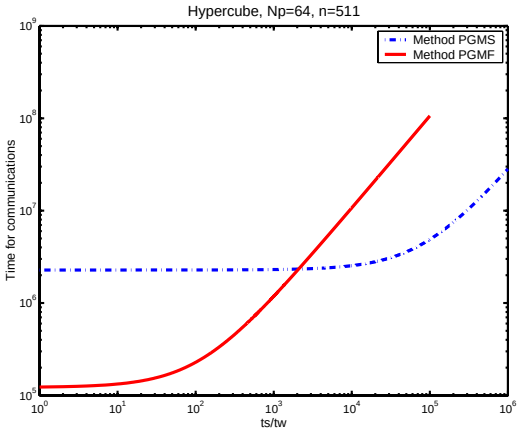
For all the cases, T_w depends on both the number of processors and the problem size. The behaviour of T_w for $m = n = 1023$ is illustrated in Figure 3 – the line for T_w^{pgms} is always above the other one.

The main advantage of the PGMS is that the number of stages at which communications are needed is a constant (4 in all) and they do not depend on the program execution, although the total amount of the transferred words is larger than that for PGMF. At each step of the forward and the backward recurrence of algorithm PFASV (i. e. Step 2 of PGMF) we have local communications. The drawback here is that global communications for groups of the processors are also performed at some of the steps depending on the size of the problem and the number of processors. The size of these groups and the amount of data depend again on mn and N_p. They are dynamically determined, which takes additional time and decreases the efficiency.

The total execution time is affected as well by parameters t_s and t_w of the particular parallel computer ($t_s \gg t_w > t_a$). In Figures 4 and 5 is presented T_{com} for hypercube as a function of $\tau = \dfrac{t_s}{t_w} \in [1, 10^6)$ for $n = 511, 1023$ and $N_p = 8, 64$. For small values of τ, T_{com}^{pgmf} grows up slowly and it is smaller than T_{com}^{pgms}. When $\tau \approx 10^2$, T_{com}^{pgmf} starts to increase faster and at some moment τ_0 it intersects the graphic of T_{com}^{pgms}. How big is τ_0 depends on the number of processors and on the size of the problem. For the presented values of n, m and N_p we have $10^3 \leq \tau_0 \leq 10^4$. Taking into account all these observations, we can conclude that on computers with small values of t_w, t_s and τ algorithm PGMF will have better performance. But on parallel systems with slow memory and communications (large t_w, t_s and τ) and especially when the processors are fast, our "expensive" direct solver GMS lead to a better parallel algorithm than the "cheaper" GMF with respect to both the measured time and the achieved speed-up.

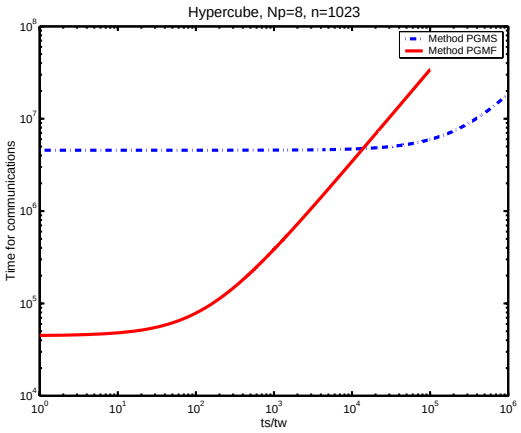


a) $N_p = 8$;

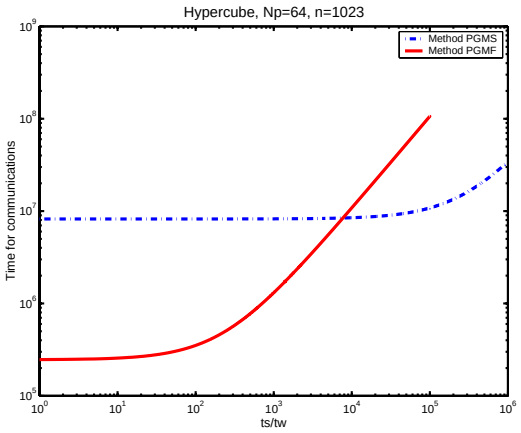


b) $N_p = 64$;

Figure 4: Communication times as a function of $\frac{t_s}{t_w}$ for $k = 7$, $n = 511$.



a) $N_p = 8$;



b) $N_p = 64$;

Figure 5: Communication times as a function of $\frac{t_s}{t_w}$ for $k = 7$, $n = 1023$.

4 Concluding Remarks

We derived in this paper theoretical estimates for the execution times of two parallel implementations (PGMF and PGMS) of the generalized marching algorithm. The presented analysis shows that each of PGMF and PGMS has its advantages and disadvantages to be kept in mind when the study is aimed in a qualitative parallel implementation on a given machine. The PGMF has smaller operation count and data to be transferred, but the number of stages for communications depend on the number of processors and the size of the problem. The number of stages for communications of PGMS is constant, but PGMS is expensive with respect to the computation count and the total amount of transferred words. On machines with different characteristics they have different performance. This is confirmed by the presented in [5] numerical results on two of the coarse-grained parallel architectures available in Texas A&M University – Silicon Graphics Origin 2000 and a Beowulf cluster of Digital (Compaq) Personal Workstations.

Future steps to complete the present study are: a) MPI/OpenMP modification of the code; b) more experiments on shared memory, distributed memory and heterogeneous systems; c) comparison of the theoretical estimates and the new numerical results. Very important are generalizations of the considered solvers to 3D case and development of efficient sequential and parallel preconditioners.

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., *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.

[5] Bencheva, G., *Parallel performance comparison of three direct separable elliptic solvers,* Large-Scale Scientific Computing, (S. Margenov, J. Wasniewski, P. Yalamov, eds.), Springer LNCS, to appear, (Extended version: Technical Report ISC-03-02-MATH `(http://www.isc.tamu.edu/iscpubs/0302.ps)`.)

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

[7] Kuznetsov, Y., Matsokin, A.M., *On partial solution of systems of linear algebraic equations,* Vychislitel'nye Metody 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.

[8] Kumar, V., Grama, A., Gupta, A., Karypis, G., *Introduction to parallel computing: design and analysis of algorithms,* Benjamin-Cummings Addison-Wesley Publishing Company,Inc., (1994).

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

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

[11] 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.

[12] Vassilevski, P.S., *An optimal stabilization of marching algorithm,* Compt. rend. de l'Acad. bulg. Sci. **41** (1988) No 7, 29–32.

[13] 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.