Fachbereich Mathematik

Matthias Pester, Sergej Rjasanow

A Parallel Version of the Preconditioned Conjugate Gradient Method for Boundary Element Equations

Preprint-Reihe der Chemnitzer DFG-Forschergruppe "Scientific Parallel Computing"

SPC 93_2 **June, 1993**

A Parallel Version of the Preconditioned Conjugate Gradient Method for Boundary Element Equations

MATTHIAS PESTER

Department of Mathematics, Technical University of Chemnitz-Zwickau,

D-09009 Chemnitz, Germany

SERGEJ RJASANOW

Department of Mathematics, University of Kaiserslautern,

D-67653 Kaiserslautern, Germany

ABSTRACT

The parallel version of precondition techniques is developed for matrices arising from the Galerkin boundary element method for two-dimensional domains with Dirichlet boundary conditions. Results were obtained for implementations on a transputer network as well as on an nCUBE-2 parallel computer showing that iterative solution methods are very well suited for a MIMD computer. A comparison of numerical results for iterative and direct solution methods is presented and underlines the superiority of iterative methods for large systems.

Keywords: boundary value problem, boundary element method, iterative method, preconditioning, parallel algorithm

1. Introduction

The boundary element method (BEM) leads to an algebraic system of linear equations with a full dense matrix [1],[26]. Such a system can be solved efficiently using preconditioned iterative methods [14],[16],[19], especially the conjugate gradient method [9],[18]. The number of arithmetical operations is $O(h^{-2})$ in two-dimensional case, where h denotes the discretization parameter in one space direction.

Such algorithms involve a very high level of parallelism which can be used by a MIMD parallel computer. The typical steps of using the BEM are: generation of the system of linear equations (matrix and right hand side) and solving this system numerically, where each step of iteration requires one matrix-vector multiplication with a full dense matrix, some scalar products, some vector additions and the solution of the preconditioning system. Each of these operations is very well-suited

for implementation on MIMD machines (such as transputer systems or the nCube machine).

Only few authors have considered the implementation of boundary element methods on parallel computers. Symm [25] and Davies [5],[6] studied the feasibility of SIMD and MIMD computers for BEM. It was shown that both the SIMD and MIMD architectures are well-suited to boundary element analysis, especially for generating the system of equations and calculating internal function values (field recovery). General problems in using direct solvers on parallel computers were discussed by Saad [21],[22]. The feasibility of SIMD and MIMD computers for solving BEM systems of linear equations numerically using the Gaussian elimination algorithm was tested by Georgiev [7] and Kreienmeyer [11]. Langer [12] deals with the parallel iterative solution of coupled FEM and BEM equations for MIMD computer with respect to domain decomposition methods.

In Section 2 we formulate the boundary value problem as well as the corresponding boundary integral equation. We discuss the parallel Galerkin discretization of the boundary integral equation in the third section. Section 4 deals with the preconditioned conjugate gradient method and the feasibility of MIMD computers for this method. Finally we present some results of our numerical experiments on parallel computers comparing iterative and direct methods.

2. Boundary integral equation

In this paper we consider the Dirichlet problem for Laplace equation in a two-dimensional bounded, simply connected domain Ω:

$$\left\{ \begin{array}{rcll} \Delta u(x) & = & 0 & , \quad x \in \Omega \\ u(x) & = & g(x) & , \quad x \in \Gamma = \partial\Omega. \end{array} \right. \tag{1}$$

If we denote by $v(x)$ the normal derivative of the function $u(x)$

$$v(x) = \frac{\partial u(x)}{\partial n(x)}\,, \; x \in \Gamma$$

then the well-known Green's third identity for potential theory

$$\frac{1}{2}u(y) = -\int_\Gamma u(x) \cdot \frac{\partial u^*(x,y)}{\partial n_x} \mathrm{d}s_x + \int_\Gamma v(x) u^*(x,y) \mathrm{d}s_x \tag{2}$$

can be used for the numerical solution of the problem (1). Here

$$u^*(x,y) = -\frac{1}{2\pi} \ln |x-y|$$

denotes the fundamental solution of the Laplace equation. The identity (2) can be written in the form

$$(\mathcal{A}v)(y) = f(y), \quad y \in \Gamma\,, \tag{3}$$

where $\mathcal{A}$ is the single layer potential operator

$$(\mathcal{A}v)(y) = \int\limits_{\Gamma} u^*(x,y)v(x)\mathrm{d}s_x$$

and

$$f(y) = \frac{1}{2}u(y) + \int\limits_{\Gamma} \frac{\partial u^*(x,y)}{\partial n_x}\mathrm{d}s_x\,.$$

The properties of the operator $\mathcal{A}$ are well-known [3],[4]:

$$\mathcal{A} : \mathbb{H}^s(\Gamma) \to \mathbb{H}^{s+1}(\Gamma)\,; \tag{4}$$

$$\langle \mathcal{A}u, v\rangle_0 = \langle u, \mathcal{A}v\rangle_0\,, \quad \forall u, v \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)\,, \tag{5}$$

$$\langle \mathcal{A}u, u\rangle_0 \geq \gamma \left\|u\right\|^2_{-\frac{1}{2}}\,, \quad \gamma > 0,\ \forall u \in \mathbb{H}^{-\frac{1}{2}}(\Gamma)\,, \tag{6}$$

where $\mathbb{H}^s(\Gamma)$ is the Sobolev space on boundary Γ, $\langle\cdot,\cdot\rangle_0$ denotes the duality pairing between $\mathbb{H}^s$ and $\mathbb{H}^{-s}$, i. e. the $L_2(\Gamma)$ inner product

$$\langle u, v\rangle_0 = \int\limits_{\Gamma} u(x)v(x)\mathrm{d}s_x$$

and $\left\|\cdot\right\|_s$ denotes the Sobolev norm in $\mathbb{H}^s(\Gamma)$.
Using the parametrization of Γ by 1-periodic representation

$$\Gamma = \left\{x \in \mathbb{R}^2 :\ x = x(t),\, 0 \leq t < 1,\ |\dot{x}(t)| \geq \delta > 0\right\},$$

we rewrite equation (3) in the form

$$(\mathcal{A}v)(\tau) = \int\limits_0^1 u^*\left(x(t), x(\tau)\right) v(t)\mathrm{d}t = f(x(\tau)), \qquad 0 \leq \tau < 1 \tag{7}$$

where $v(t) := v(x(t)) \cdot |\dot{x}(t)|$. æ

3. Galerkin method

We begin with the approximation of $v(t)$ by piecewise B-splines

$$\varphi_l^{(\nu)}(t),\ l = 1, \ldots, N$$

of degree $\nu = 0, 1, 2$. We divide the interval $[0, 1)$ into $N > \nu + 1$ subintervals

$$[0,1) = \bigcup_{l=1}^{N} [t_l, t_{l+1})\,, \quad t_l = (l-1)h,\ h = \frac{1}{N},$$

and introduce the N-dimensional subspace $\mathbb{H}_N$ of 1-periodic functions (see [19])

$$\mathbb{H}_N = \operatorname{span}\left(\varphi_1^{(\nu)}(t), \ldots, \varphi_N^{(\nu)}(t)\right).$$

The Galerkin method for equation (7) leads to:

Finding the function $v_h(t) \in \mathbb{H}_N$ such that Galerkin equations

$$\langle \mathcal{A} v_h, w \rangle_0 = \langle f, w \rangle_0 \tag{8}$$

are satisfied for all functions $w \in \mathbb{H}_N$.

Equation (8) is equivalent to the following system of linear equations:

$$Ay = b, \quad A \in \mathbb{R}^{N\times N},\ y, b \in \mathbb{R}^N, \tag{9}$$

where the elements a_{ij} and b_i are of the following form:

$$\begin{aligned} a_{ij} &= \left\langle \mathcal{A}\varphi_j^{(\nu)}, \varphi_i^{(\nu)} \right\rangle_0 = \int_0^1 \int_0^1 u^*(x(t), x(\tau))\varphi_j^{(\nu)}(t)\varphi_i^{(\nu)}(\tau)\mathrm{d}t\mathrm{d}\tau, \\ b_i &= \left\langle f, \varphi_i^{(\nu)} \right\rangle_0 = \int_0^1 f(x(\tau))\varphi_i^{(\nu)}(\tau)\mathrm{d}\tau. \end{aligned} \tag{10}$$

The numerical computation of matrix A and of the right hand side b is one of the most important and time consuming steps in the realization of BEM. The numerical integration is necessary for the evaluation of integrals in (10). The following idea is very useful for this purpose (see [10], [19]):

We split operator $\mathcal{A}$ of equation (3) as a sum of two operators $\mathcal{A}_1$ and $\mathcal{A}_2$ with

$$\left(\mathcal{A}_1 v\right)(\tau) = -\frac{1}{2\pi}\int_0^1 \ln\left|\sin\pi(t-\tau)\right| v(t)\mathrm{d}t,$$

$$\left(\mathcal{A}_2 v\right)(\tau) = -\frac{1}{2\pi}\int_0^1 \ln\frac{\left|x(t)-x(\tau)\right|}{\left|\sin\pi(t-\tau)\right|} v(t)\mathrm{d}t\ .$$

The kernel of operator $\mathcal{A}_2$ is continuous for all t and τ:

$$\lim_{t\to\tau}\left(-\frac{1}{2\pi}\ln\frac{\left|x(t)-x(\tau)\right|}{\left|\sin\pi(t-\tau)\right|}\right) = -\frac{1}{2\pi}\ln\left|\dot{x}(\tau)\right|\ ,$$

and operator $\mathcal{A}_1$ describes the principal part of operator $\mathcal{A}$. The matrix of system (9) can also be splitted as a sum of two matrices $A_1^{(\nu)}$ and $A_2^{(\nu)}$

$$\left(A_\alpha^{(\nu)}\right)_{ij} = \left\langle \mathcal{A}_\alpha \varphi_j^{(\nu)}, \varphi_i^{(\nu)} \right\rangle_0, \quad \alpha = 1, 2\,,$$

where the elements of matrix $A_1^{(\nu)}$ can be computed analytically (A_1 is independent of the special choice of boundary Γ) and the elements of matrix $A_2^{(\nu)}$ are computed as follows.

Let $Q \in \mathbb{R}^{N\times N}$ denote the matrix with the elements

$$q_{kl} = u^*(x(t_k), x(t_l)) - \frac{1}{2\pi} \ln|\sin(\pi(t_k - t_l))|, \quad k,l = 1,\ldots,N,$$

and $f \in \mathbb{R}^N$ denotes the vector with the components

$$f_k = f(x(t_k)), \quad k = 1,\ldots,N.$$

We compute the approximations for matrix $A_2^{(\nu)}$ and vector b in (9) as follows (see [10],[19]):

$$A_2^{(\nu)} = h^2 Q, \quad b = h \cdot f, \qquad \text{if} \quad \nu = 0, \tag{11}$$

$$\begin{aligned} A_2^{(\nu)} &= h^2 \left[(\omega_0^{(\nu)} Q + \omega_0^{(\nu)}\omega_1^{(\nu)}(QJ + QJ^\top + JQ + J^\top Q) + \right. \\ &\quad \left. + (\omega_1^{(\nu)})^2 (JQJ + J^\top QJ + JQJ^\top + J^\top QJ^\top) \right] \\ b &= h(\omega_0^{(\nu)} f + \omega_1^{(\nu)}(Jf + J^\top f), \qquad \text{if} \quad \nu = 1,2, \end{aligned} \tag{12}$$

where the weights $\omega_i^{(\nu)}$, $i = 0,1$ are given by:

$$\omega_0^{(1)} = \frac{5}{6}, \qquad \omega_1^{(1)} = \frac{1}{12}$$

$$\omega_0^{(2)} = \frac{3}{4}, \qquad \omega_1^{(2)} = \frac{1}{8}$$

and J denotes the simplest circulant matrix

$$J = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Obviously, the computations of (11) and (12) can be done parallelly where each processor has to compute and store only a strip of rows (or columns) of matrix A and the corresponding number of elements of the right-hand side vector b as shown in Figure 1. Matrix Q depends on the boundary information which is kept completely in the local memories of each processor, after an initial broadcast. This little amount of redundance leads to a full parallel computation of Q which requires no interprocessor communication.

We assume that each processor p $(p = 0,\ldots,P-1)$ has computed a block $Q^{(p)}$ of rows of the full matrix Q. In the case of $\nu = 0$, matrix A can be obtained directly from Q using only the locally stored elements on each processor. For $\nu = 1$ or $\nu = 2$, there are two ways of computing A. The distributed application of matrix J to matrix

Q requires the last row of block $Q^{(p-1)}$ computed by processor $p-1$ and the first row of block $Q^{(p+1)}$ computed by processor $p+1$ (where "$p-1$" and "$p+1$" are considered to be operations $\mathbf{mod}(P)$ for a ring numbering of the P processors).
The current processor p might

(i) compute either two rows of Q in addition to the local matrix block $Q^{(p)}$ or

(ii) exchange its first row with the last row of processor $p-1$ and its last row with the first row of processor $p+1$.

Vector b is treated in the same way having single elements instead of rows (Figure 1). The efficiency of the method depends on the facilities of the actual parallel computer's hard- and software. So, the processor ring is appropriate for method (ii), because data exchanges occur between neighboring processors only.

In our application we preferred the hypercube topology with respect to the later solution of the system of equations. Although there is always an embedded ring in a hypercube (see [23],[15]), we did not use this ring sequence to place the matrix blocks $Q^{(p)}$ at the processors in order to keep some other helpful properties of the original numbering in the hypercube. Thus two processors p and $p+1$ need not be direct neighbors, and it may be better to use method (i) mentioned above if we do not want to spend much time on a message routing system. Currently, however, the concept of virtual channels based on hardware routing will enable both the hypercube and the ring topology without permutations of processor numbers.

Figure 2 shows computation time (in seconds) for parallel generating matrix A and right-hand side vector b for different problem size N and increasing number of processors ($p = 1, \ldots, 128$). These results were obtained on a GC-el system of T805 transputers, and by analogy on nCube-2 with a factor of about 0.5 in actual times but with the same behavior in general.

The figure shows an optimal scale-up for both increasing problem size and number of processors. Considering the lower right corner of the figure, there is visible the overhead that occurs for a too small problem size with respect to the large number of processors.

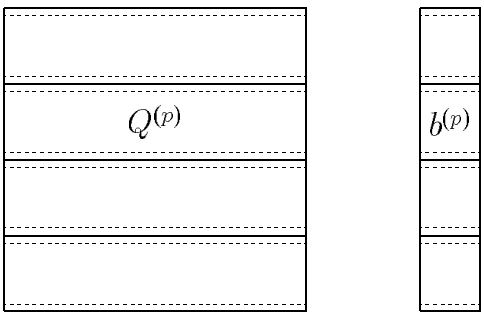

Fig. 1: Data placement on processors.

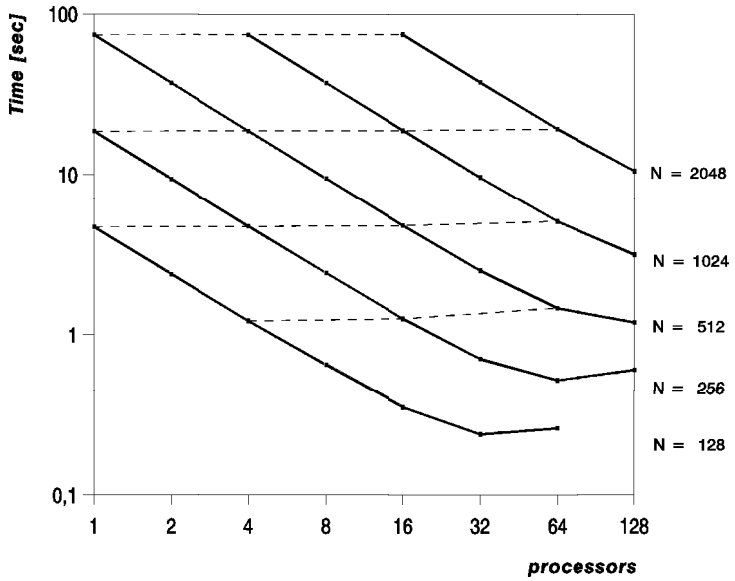

Fig. 2: Time for parallel generating the BEM system matrix.
Dotted lines show a very good scale-up.

4. Preconditioned conjugate gradient method

One of the most effective iterative solution methods for the symmetric, positive definite system of linear equations

$$Ay = b, \qquad A \in \mathbb{R}^{N\times N}, \quad y, b \in \mathbb{R}^N, \quad A = A^\top > 0 \tag{13}$$

is the conjugate gradient method [9], where $B = B^\top > 0$ is a preconditioning matrix.

1. **Initial step:**
$$\begin{aligned} y_0 &\in \mathbb{R}^N; \\ r_0 &= Ay_0 - b; \\ w_0 &= B^{-1}r_0; \\ s_0 &= w_0; \end{aligned}$$

2. **repeat for** $k = 0, 1, 2, \ldots$

$$\begin{aligned} y_{k+1} &= y_k - \alpha_{k+1}s_k, & \alpha_{k+1} &= \frac{(r_k, w_k)}{(As_k, s_k)}; \\ r_{k+1} &= r_k - \alpha_{k+1}As_k; \\ w_{k+1} &= B^{-1}r_{k+1}; \\ s_{k+1} &= w_{k+1} + \beta_{k+1}s_k, & \beta_{k+1} &= \frac{(r_{k+1}, w_{k+1})}{(r_k, w_k)}, \end{aligned}$$

In order to use this method for our system (9), we have to prove that matrix A is symmetric and positive definite. The symmetry follows from the property (5) of operator $\mathcal{A}$. Matrix A is positive definite, because of property (6), and matrix A has the spectral condition number

$$\kappa(A) = O(h^{-1}) \tag{14}$$

because of property (4). The speed of convergence of CG method depends on the spectral condition number of the matrix $B^{-1}A$. The optimal preconditioning matrix B for the discrete single layer potential is well-known (see [26],[14],[16]): it is the Galerkin matrix B for operator $\mathcal{B}$ defined by

$$(\mathcal{B}u)(\tau) = -\frac{1}{2\pi}\int_0^1 \ln\left|\sin\pi(t-\tau)\right| u(t)\mathrm{d}t\,.$$

i.e. $\mathcal{B}$ is equal to $\mathcal{A}_1$ and $B = A_1^{(\nu)}$. The spectral condition number of matrix $B^{-1}A$ is bounded:

$$\kappa(B^{-1}A) = O(1). \tag{15}$$

The bound is independent of h, and the number of iterations would also be independent of h. One step of iterations requires the matrix-vector multiplication with a full dense matrix (As_k), two scalar products $((As_k, s_k)$ and $(r_{k+1}, w_{k+1}))$, three vector additions and the solution of the preconditioning system

$$Bw_{k+1} = r_{k+1}\,. \tag{16}$$

A very efficient method for the numerical solution of system (16) arises from the special properties of matrix B. This matrix is symmetric, positive definite and circulant. It can be written in the form

$$B = N^{-1}F\Lambda F^*\,, \tag{17}$$

where F denotes the matrix of discrete Fourier transform

$$f_{kl} = \boldsymbol{e}^{i\frac{2\pi}{N}(k-1)(l-1)}, \qquad k,l = 1,\ldots,N$$

and Λ is a diagonal matrix with the eigenvalues of B:

$$\Lambda = \mathrm{diag}(\beta_1,\ldots,\beta_N) = \mathrm{diag}(FB^{\top}\boldsymbol{e}_1)\,. \tag{18}$$

Here, $\boldsymbol{e}_1 = (1,0,\ldots,0)^{\top}$ denotes the first column of the unit matrix. The solution w_{k+1} of the system (16) can be given with the help of (17) as

$$w_{k+1} = B^{-1}r_{k+1} = N^{-1}F\Lambda^{-1}F^*r_{k+1} \tag{19}$$

and computed using the Fast Fourier Transform (FFT) [2],[8].

It is necessary to remark at this place that we are considering only a model problem (1). The algorithms with circulant matrices seem to be of importance for many other problems arising from BEM:

- We can use $\mathbf{diag}(B, B)$ as a preconditioning matrix for the Dirichlet problem of linear elasticity in two-dimensional case.

- The problem with Neumann boundary condition can be solved analogously with the help of hypersingular operator (see [3]) where the optimal preconditioner is a circulant, too.

- The problem in a multiple connected domain leads to a system of linear equations with a block system matrix. The matrix $\mathbf{diag}(B_1, \ldots, B_k)$ is an optimal preconditioner for such a problem, where B_i denotes the matrix arising from $A_1^{(\nu)}$ for boundary Γ_i (see [20]).

- The three-dimensional Dirichlet problem on a rotational domain leads to a so called circulant-block matrix (all blocks are circulant matrices, see [13],[17]). Furthermore, the algorithms with circulant matrices presented in [17] are very suitable for parallel computers.

- The optimal preconditioner based on the theory of circulant matrices is also very useful for constructing a global preconditioner for coupled BEM-FEM discretization for nonlinear problems with the domain decomposition (see [12]).

5. Parallel solution and numerical results

We consider the system of equations (13) to be solved on a parallel computer using the preconditioned CG method as described above. In order to take full advantage of the Fast Fourier Transform as an efficient preconditioner, we assume a problem size of $N = 2^m$. The parallel computer should be a hypercube of $P = 2^n$ general purpose processors (our test computers were microprocessor systems based on transputers or nCube-2). Generally, we have $n < m$, i.e. $P \ll N$.

Then the single processor p works with a block $A^{(p)}$ of $N_p = \frac{N}{P}$ rows of matrix A and the corresponding parts of all the vectors needed for the CG algorithm. Thus, we have a global algorithm working on multiple processors with different parts of data. That means, each of the basic operations of CG method should be done in parallel (see [15]):

5.1. *Vector additions*

The operations of the kind $y := y - \alpha s$ (*"DAXPY"*) can be completely executed in parallel by computing the local part $y^{(p)} := y^{(p)} - \alpha s^{(p)}$ on each processor p.

5.2. *Scalar products*

The computation of the inner product of two vectors is split into P local partial sums for each processor p and a global sum of those numbers over all processors

$$(r, w) = \sum_{i=1}^{N} r_i w_i = \sum_{p=0}^{P-1} \left(\sum_{i=1}^{N_p} r_i^{(p)} w_i^{(p)} \right)$$

requiring only n steps of next-neighbor communication within the n-dimensional hypercube.

5.3. *Matrix-vector multiplication*

We assume Matrix A to be generated as described in Section 3, i. e. processor p (for $p = 0, \ldots, P-1$) has to compute and store in its local memory the block $A^{(p)}$ of N_p rows of the matrix (row numbers $p \cdot N_p + 1, \ldots, (p+1) \cdot N_p$). Now we treat $A^{(p)}$ as an array of P $N_p \times N_p$ square matrix blocks $A_j^{(p)}$ $(j = 0, \ldots, P-1)$.

The matrix-vector multiplication can be performed block-wise with high efficiency by communicating via the processor ring embedded in the hypercube (Figure 3). The local computation is partitioned into P consecutive steps, each of them containing three (*parallel executable*) operations:

- **compute** $y^{(p)} = y^{(p)} + A_j^{(p)} x^{(j)}$
- **send** j and $x^{(j)}$ to the forward ring neighbor
- **receive** j' and $x^{(j')}$ from the backward ring neighbor

where the meaning of j and j' is exchanged after each step. A comparison of different implementation methods for this operation is given in [15]. Best results we got on a transputer system with a standalone Fortran compiler (i. e. without operating system) where the **send** and **receive** tasks were implemented as parallel subthreads to get full hardware performance. In this case the communication time could be neglected. Otherwise (sequential operating on each processor) we have to accept the communication complexity of $2N$ vector elements caused by sending vector x through the processor ring in $2P$ operations (*send/receive*).

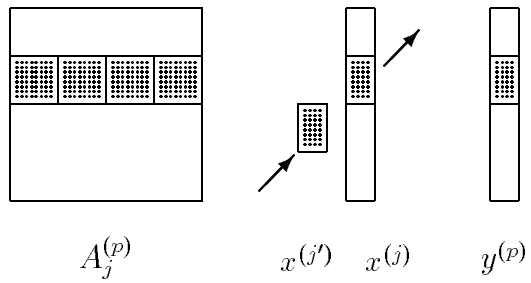

Fig. 3: Block-wise piped matrix-vector multiplication

5.4. *FFT - preconditioning*

The well-known FFT algorithm to compute $y = F \cdot x$ for a vector x of length $N = 2^m$ consists in executing m steps of *butterfly operations* (Figure 4) of the kind

$$y_j^{(s)} = f_l y_{j1}^{(s+1)} + y_{j0}^{(s+1)}$$

where s denotes the decreasing step number ($s = m-1, \ldots, 0$; $y^{(m)} = x$; $y^{(0)} = y$) and

$$\begin{aligned} j1 &= j_{m-1} \ldots j_{s+1} 1 j_{s-1} \ldots j_0 = \mathsf{OR}(j, 2^s) \\ j0 &= j_{m-1} \ldots j_{s+1} 0 j_{s-1} \ldots j_0 = j1 - 2^s \\ l &= j_{m-1} \ldots j_s = \left[j \cdot 2^{-s} \right] \end{aligned}$$

The *butterfly* operation itself is performed by computing both $y_{j0}^{(s)}$ and $y_{j1}^{(s)}$ (for

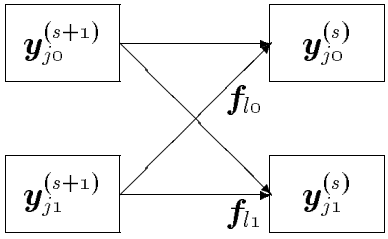

Fig. 4: Single FFT butterfly step.

$j = j0$ and $j = j1$). The coefficients f_l are always the same just for a sub-vector of 2^s elements. Thus, the butterfly operations are considered to be executed on sub-vectors instead of single elements. In this way, the first step of FFT is only *one* butterfly operation for two vectors of length $\frac{N}{2}$, and the last step is executed by $\frac{N}{2}$ butterfly operations with pairs of single elements (see Figure 5). This binary structure of the algorithm is appropriate for parallel implementation on a hypercube. If we have stored the local part $y^{(p)}$ of the vector y on processor p, then the global index $j = j_{m-1} \ldots j_0$ ($j_i \in \{0, 1\}$) of any element contains the processor number p in its first n binary digits and the local index j_{loc} in its last $m - n$ digits:

$$p = j_{m-1} \ldots j_{m-n}, \qquad \text{and} \qquad j_{loc} = j_{m-n-1} \ldots j_0.$$

Only a small amount of next-neighbor communication appears within the first n steps where processors have to exchange their local vectors. The last $m - n$ steps are executed independently on all processors with full parallelism and a speed-up which is close to the optimum if $(m - n) \gg n$.

Generally the single Fourier transform by FFT requires some reordering of vector elements using the *bit-swapped* index as shown in Figure 5 (rightmost column). Such

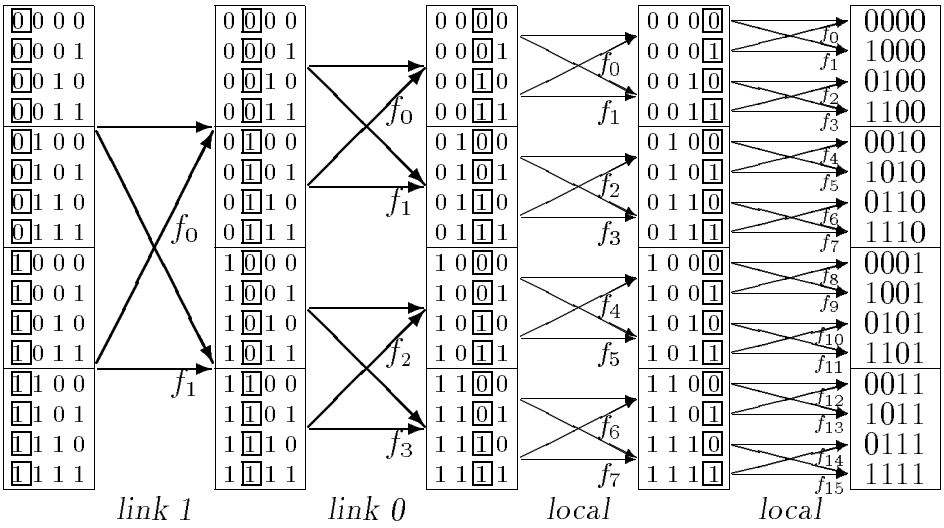

Fig. 5: Data flow for FFT on 4 processors
The emphasized bit of the index value determines the associated index for the next butterfly operation.

a reordering of a distributed vector would be very inefficient because of the global character of data exchange. On the other hand, in order to apply the preconditioning operator B^{-1} from (19) we have to execute consecutively:

(i) a Fourier transform $y^{(1)} := F^* x$, which is executed as described above;

(ii) a simple element-wise vector multiplication $y^{(2)} := \Lambda^{-1} y^{(1)}$ and

(iii) a second Fourier transform $y := F y^{(2)}$.

Within the intermediate step (ii) we have two vectors (diag(Λ) from equation (18) and $y^{(1)}$ from step (i)) both obtained by FFT and therefore having the same bit-swapped index ordering.

Hence, there is no reason to reorder the vector elements which would require an essential amount of global processor communication. The only thing we have to do is to implement step (iii) in another way different from step (i), using the reversed data flow and corresponding coefficients, i. e. execute the *first* $m-n$ steps locally, and only the *last* n steps include next-neighbor communication. Thus, the preconditioning operator has a communication complexity of $4n\frac{N}{P}$ vector elements which results from executing the FFT algorithm twice with $4n$ single operations (*send/receive*) in all.

Now, the total communication complexity of one step of preconditioned CG iteration can be determined as a function of the problem size N and the number of processors $P = 2^n$:

$$2N + 4n\left(1 + \frac{N}{P}\right)$$

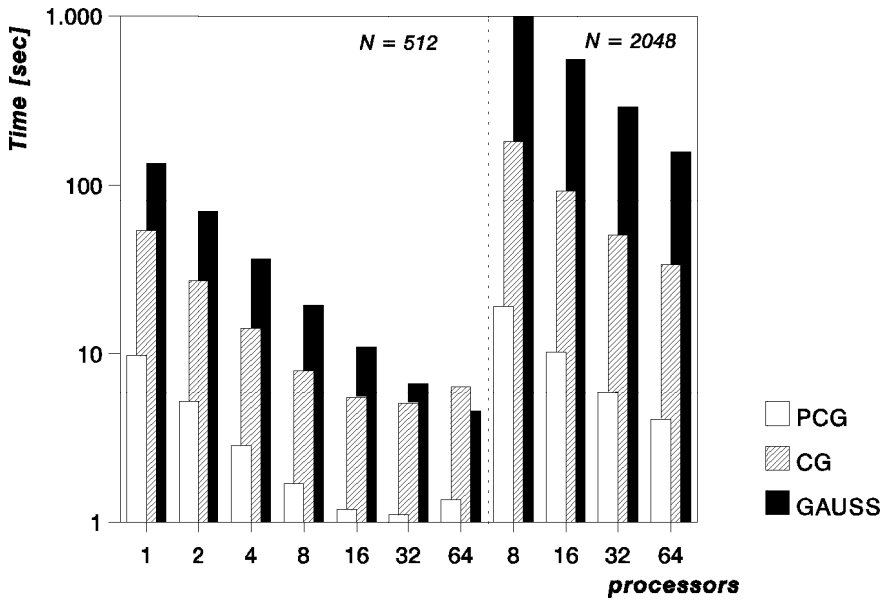

Fig. 6: Comparing times for iterative and direct solvers for dense linear systems on a parallel computer (computation on nCube-2).

obviously increasing only with N and $\log P$.

Due to the constant number of iterations for a given problem (independent of N) caused by the $O(1)$-property (15), we can state a similar relation for the complete PCG algorithm. The real behavior, however, depends substantially on the bandwidth of the current communication network and on the relation of the start-up and transmitting times of the message passing system. In our implementation one step of Preconditioned CG iteration requires $8n + 2P$ communication steps (i.e. start-ups) on each processor. Surely, there is an overhead of communication if the problem size is too small for the number of processors in use. So, for example, we find a minimum of total computation time at 32 processors for a problem size $N = 512$.

The communication complexity of the pipelined ring algorithm for Gaussian elimination does not depend on the number of processors. This result of Saad [21] was certified once more by our numerical experiments. With respect to the large amount of processor time needed for this direct solver there is a very good speed-up within reach. The absolute computation time of this direct solver (processor time plus communication time), however, extremely exceeds that of the preconditioned iterative method.

Figure 6 illustrates some results of our numerical experiments for parallel solving the dense linear system (13), comparing the *preconditioned CG algorithm* with a simple *CG method* and the *pipelined ring algorithm of Gaussian elimination* as proposed by Saad [22].

We should remark that similar results were obtained on different parallel computers of MIMD type (transputer, nCube) and also, with substantially higher portion of

Table 1: Actual times and percentage of communication for solving dense linear systems of maximum size N on P processors by different methods (T805 transputer, 4 MByte RAM each)

		Gaussian elimination		simple CG algorithm		preconditioned CG	
P	N	total	I/O	total	I/O	total	I/O
1	512	202 s	—	78 s	—	15.0 s	—
4	1024	416 s	3.7 %	103 s	2.1 %	15.3 s	5.4 %
16	2048	855 s	7.7 %	135 s	4.5 %	15.8 s	7.5 %
64	4096	1839 s	15.1 %	196 s	12.4 %	18.4 s	15.0 %

communication, on a workstation cluster using PVM [24].

In order to evaluate the behavior of parallelized algorithms, absolute times are of less interest than the relations between them. The most valuable result is given by comparing the time for solving a system of the maximum size that is possible for any given number of processors with a given memory size (Table 1). A nearly constant time for such a scale-up indicates the very good suitability of the preconditioned CG method for parallel computers.

References

[1] C. A. Brebbia and S. Walker. *Boundary Element Techniques in Engineering.* Newnes–Butterwords, (1980).

[2] J. W. Cooley and J. W. Tukey. *An algorithm for the machine calculation of complex Fourier series.* Math. Comput., **19** (1965) 297–301.

[3] M. Costabel. *Principles of boundary element methods.* Comp. Phys. Reports, **6** (1987) 243–274.

[4] M. Costabel and W. L. Wendland. *Strong ellipticity of boundary integral operators.* J. Reine Angew. Math., **372** (1986) 34–63.

[5] A. J. Davies. *The boundary element method on a transputer network.* in C. Brebbia and G. Gipson (editors), Boundary Elements XIII, pages 985–994. Computational Mechanics Publications, Southampton Boston, (1991).

[6] A. J. Davies. *The implementation of the boundary element method on a network of transputers.* Technical report 241, Hatfiled Polytechnical NOC, (1991).

[7] K. Georgiev and S. Margenov. *Boundary element method realization on a computing system including a systolic processor.* Technical report, Center of Informatics and Computer Technology, Bulgaria, (1988).

[8] P. Henrici. *Fast Fourier methods in computational complex analysis.* SIAM Rev. **21** (1979) 481–527.

[9] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems.* J. Res. NBS **49** (1952).

[10] G. C. Hsiao, P. Kopp, and W. L. Wendland. *A Galerkin collocation method for some integral equations of the first kind.* Computing **25** (1980) 89–130.

[11] M. Kreienmeyer. *Zur Parallelisierung der 2D–BEM unter ParC.* Lecture on a workshop on Boundary Element Methods in Chemnitz, Universität Hannover, (May 1992).

[12] U. Langer. *Parallel iterative solution of symmetric coupled FE/BE-equations via domain decomposition.* Preprint 217, TU Chemnitz (1992).

[13] A. Meyer and S. Rjasanow. *An effective direct solution method for certain boundary integral equations in 3D.* Math. Methods in Appl. Sciences, **13** (1990) 45–53.

[14] Y. N. Mokin. *Direct and iterative methods for solving the integral equations of potential theory.* U.S.S.R. Comput. Maths. Math. Phys., **28** (1989) 29–39.

[15] M. Pester. *Implementation und Test paralleler Basisalgorithmen der linearen Algebra,* in R. Grebe and C. Ziemann (editors), Parallele Datenverarbeitung mit dem Transputer. 2. Transputer–Anwender–Treffen TAT'90, Proceedings X, Informatik–Fachberichte, vol. 272, pp. 111–118. Springer–Verlag, (1991).

[16] L. Reichel. *A method for preconditioning matrices arising from linear integral equations for elliptic boundary value problems.* Computing, **37** (1986) 123–136.

[17] S. Rjasanow. *Effective algorithms with circulant-block matrices.* to appear in: Linear Algebra Appl.

[18] S. Rjasanow. *CG-Verfahren für die Randintegralgleichungen.* Preprint 61, TU Karl-Marx-Stadt, (1988).

[19] S. Rjasanow. *Vorkonditionierte iterative Auflösung von Randelementgleichungen für die Dirichlet–Aufgabe.* Wiss. Schriftenreihe **7**, TU Chemnitz, (1990).

[20] S. Rjasanow. *Effective iterative solution methods for Boundary Element Equations,* in C. Brebbia and G. Gipson (editors), Boundary Elements XIII, pp. 889–900. Computational Mechanics Publications, Southampton, Boston, (1991).

[21] Y. Saad. *Communication complexity of the Gaussian elimination algorithm on multiprocessors.* Linear Algebra Appl., **77** (1986) 315–340.

[22] Y. Saad. *Gaussian elimination on hypercubes,* in Proc. Int. Workshop, pp. 5–17. Luminy/France, (1986).

[23] Y. Saad and M. H. Schultz. *Topological properties of hypercubes.* Research Report 389, Yale University, Dept. Computer Science, (1985).

[24] V. S. Sunderam. *PVM: A framework for parallel distributed computing.* Technical report, Oak Ridge National Laboratory, (1992). included in the PVM public domain package.

[25] G. T. Symm. *Boundary elements on a distributed array processor.* Engineering Analysis, **1** (1984) 162–165.

[26] W. L. Wendland. *Bemerkungen zu Randelementmethoden und ihren mathematischen und numerischen Aspekten.* Mitteilungen der GAMM, Heft **2** (1986) pp. 3–27.