PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 28/92

A Direct Domain Decomposition Procedure for Elliptic Problems

Maurício Kischinhevsky

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 28/92
Editor: Carlos J. P. Lucena Julho, 1992

A Direct Domain Decomposition Procedure for Elliptic Problems*

Maurício Kischinhevsky

* This work has been sponsored by the Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br
techrep@inf.puc-rio.br (for publications only)

A direct domain decomposition procedure for elliptic problems

Mauricio Kischinhevsky

Departamento de Computação - UFF
e-mail: gmcmkis@bruff.bitnet
Departamento de Informática - PUC-RJ
e-mail: mauricio@inf.puc-rio.br

july/1992

Abstract

This report describes a new parallel solver for linear systems of equations arising from the discretization of certain elliptic partial differential equations. A domain decomposition procedure is generated making use of the statistical solution of partial differential equations. The new solver is very well suited for implementation on a loosely coupled parallel environment. An implementation and an analysis of the algorithm proposed are described for the case a conjugate gradient solver is used on the subproblems. Numerical results confirm effectiveness of the procedure.

Keywords: domain decomposition methods, monte carlo solution of partial differential equations, parallel solution of linear systems of equations, parallel conjugate gradient method.

Resumo Este relatório descreve um novo resolutor paralelizável para sistemas de equações lineares oriundos da discretização de certas equações diferenciais parciais elíticas. Um procedimento de decomposição de domínios é gerado a partir de uma estratégia estatística de solução local de equações diferenciais parciais. O novo resolutor é bastante adequado para ambientes computacionais com paralelismo fracamente acoplado. Implementação e análise do algoritmo são apresentados para um caso em que um resolutor por gradientes conjugados é utilizado nos subproblemas.

Palavras-chave: métodos de decomposição de domínios, solução por monte carlo de equações diferenciais parciais, resolução paralela de sistemas de equações lineares, gradientes conjugados paralelos

1 Introduction

The interest in efficiently solving large scale sparse systems has led to many strategies, from iterative methods such as Gauss-Jacobi, Gauss-Seidel, SOR, steepest descent, multigrid and conjugate gradient to direct ones like gaussian elimination, Crout-Banachiewicz LU decomposition or conjugate gradient (see ,e.g., [17][1][15][4]).

The main problem is to solve

$$Hx = b\,, \quad x, b \in \mathbf{R}^N \quad , \tag{1}$$

and H is an $N \times N$ positive definite matrix, in a convenient way for the specific application.

Conjugate gradient (CG) methods (see,e.g., [2] for a review) are an efficient tool for iterative solution of large sparse sets of linear systems of equations since successive estimates of the solution vector are generated, with non-decreasing number of significant digits at each step, in the absence of rounding errors.

Preconditioning is frequently a means of accelerating the cg method, causing a reduction in the number of iterations performed, although the cost per iterative step is increased. Many preconditioning strategies are available, including, e.g., incomplete factorization, polynomial [3].

Multigrid (mg) [13][10] methods form iterative procedures which are being widely used and exhibit asymptotic convergence rates very convenient for large systems, although with higher costs per iteration.

The possibility of effectively using parallel architectures led to an increase in the search for efficient algorithms. Some old algorithms were modified to the new environments, while parallelism-oriented algorithms are also being developed. Domain decomposition methods recently developed make use of such a concept, being well suited for parallel computing.

Recalling that direct methods are not, in general, adequate for parallelization, one is left with the iterative procedures, to try the real jump towards efficiency in parallel environments. Methods of mg type are parallelizable in the interpolating and in the restriction operator implementations, although complete parallelism between different level operations is not achievable. This is due to the need of one level's result in order to carry out the next level's calculations. Moreover, cg methods also present difficulties, since its only improvement is achieved by parallelization of matrix-vector multiplications, that causes an increase in message passings[1], unless the configuration allows sharing of data (that is, strongly coupled multiprocessor systems).

Some successfull iniciatives were employed in preconditioning, that is, some pcg methods have shown to be very adequate for parallel computation (see, e.g. [9],[8]), specially in strongly coupled environments.

The domain decomposition conjugate gradient (ddcg) method thus presented is a strategy which builds an effective parallelization of any linear systems' solver (here, cg method), once it is combined with a statystical method in some subregions of the original computational domain, Ω. The strategy then makes use of a class of methods that allows the determination of local solutions of the boundary value problem (bvp) independently from the other points. Additionally, this statystical method is intrinsically very well suited for implementation on a distributed memory parallel computer, since communications between pairs of processors are not so frequent relative to the amount of local computations.

This report is organized as follows: the new algorithm is presented for elliptic problems decomposed in subregions. Next, a minimization procedure is used in the smaller problems. Then, a code is shown that illustrates the main features of the new procedure when applied to a simple model problem. Moreover, two approaches, the discrete and the continuous one, are introduced and analysed.

[1]Vectorization is naturally usable to fasten matrix-vector and vector-vector operations, accelerating cg procedures, but is not the tool this report mainly intends to focus on.

2 Conjugate gradient and domain decomposition

2.1 Some aspects of the conjugate gradient method

The conjugate gradient method [17] was proposed as a means to solve a minimization (or maximization) problem for quadratic functionals of the form [2]

$$f(x) = \frac{1}{2}x^t H x - b^t x + c, \tag{2}$$

where H is positive (or negative) definite.

In the usual notation the gradient and the Hessian of $f(x)$ are easily found to be $g(x) = Hx - b$ and $H(x) = H$, respectively. Thus, such quadratic functionals have constant Hessian.

A point $\hat{x}$ is a stationary point of f if the gradient vanishes at $\hat{x}$, that is, if $H\hat{x} - b = 0$. If H is non-singular, $\hat{x}$ is uniquely determined by $\hat{x} = H^{-1}b$.

The method of steepest descent(sd), which brings the basic concepts to cg works through a sequence of iterative descent steps. The aim at each is to minimize the value of $f(\mathrm{x})$ on the direction $-\nabla f$ (the **search direction**).

The algorithm cg is an extension of sd, and considers the possibility of combining the usual **search direction** of SD with the value at the previous step. Further details about cg methods can be found in [17],[4].

In order to obtain a solution with the required accuracy, iterations of the type below are performed. Namely,

procedure conjugate gradient(cg)
For $k = 0, 1, \ldots$ **and** x^0 **chosen, set** $g^0 = Hx^0 - b$, $d^0 = -g^0$ **and**

$$\tau_k = \frac{g^k g^k}{d^k H d^k}$$
$$x^{k+1} = x^k + \tau_k d^k$$
$$g^{k+1} = g^k + \tau_k H d^k$$
$$\beta_k = \frac{g^{k+1} g^{k+1}}{g^k g^k}$$
$$d^{k+1} = -g^{k+1} + \beta_k d^k$$

End_For
End of cg.

Some remarks are necessary in order to highlight the importance of cg methods. The most valuable properties found in these methods are the orthogonalities of two sets of vectors generated, namely g's and d's, the first in euclidean and the second in energy norms. Moreover, it can be shown ([4]-App.A) that, for any $\epsilon > 0$, if $p(\epsilon)$ is defined to be the smallest integer k such that

$$\|x^k - \hat{x}\|_H \leq \epsilon \|x^0 - \hat{x}\|_H \quad , \qquad \forall x^0 \in R^N \quad , \tag{3}$$

then

$$p(\epsilon) \leq \frac{1}{2}\sqrt{K(H)} ln(\frac{2}{\epsilon}) + 1 \tag{4}$$

where $K(H)$ is the condition number $\frac{\lambda_N}{\lambda_1}$ for matrix H.

[2] For clarity one may ommit the symbol of transposition where no confusion is possible

That is, cg method presents the finite termination property, in the absence of rounding errors, but perhaps more importantly, **low cost per iteration** and **strong dependence on initial estimate** x^0. The low cost of each iteration is evident in the algorithm, where only one matrix-vector multiplication is found inside the loop. The dependence on first estimate for x becomes particularly important in case $K(H)$ is not small.

Asymptotic estimates for convergence rates are an important tool in the analysis of algorithms, and mg methods beat cg at this point. But the presence of so many desirable properties in cg methods explain the continued interest in developing enhancements for it(see,e.g., [5]). As the convergence rate is not optimal like in mg, some efforts are oriented towards reduction of $K(H)$, via optimized numbering of discrete variables and the development of efficient preconditioners. Here it is claimed that domain decomposition is an extremely convenient framework for the maintenance of cg methods as one of the most efficient tools in the solution of discrete systems arising from a wide variety of partial differential equations, specially with adequate use of vector and parallel processing.

2.2 The ddcg strategy

2.2.1 A simple model problem and discrete random walks

When the solution of a pde is required in few points of the domain, monte carlo methods can be employed successfully. Here a small review of this concept is presented (for further details see,e.g. [16][12]) to set notation and terminology.

Consider Laplace's equation inside Ω (unit square) with Dirichlet boundary data. That is

$$\nabla^2 u = 0, \ in \ \Omega \tag{5}$$

and

$$u = g(\mathbf{x}) \, , \mathbf{x} \in \partial\Omega \, . \tag{6}$$

Noting that the simple five point discretization of the laplacian leads to

$$\begin{aligned} u_{i,j} &= \frac{1}{4} \cdot (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \quad (i,j) \text{ an interior point} \\ u_{i,j} &= g_{i,j}, \quad g_{i,j} \text{ the solution at a boundary point } (i,j) \, , \end{aligned} \tag{7}$$

one can perform an analogy with a random walk starting at (i,j). In fact suppose an award is given (value $g_{i,j}$) each time the walk is to start (and instantaneously end) at a boundary point (i,j). Moreover, for any interior point let the score be given by an average of the neighboring values, that is,

$$R(A) = \frac{1}{4}[R(B) + R(C) + R(D) + R(E)], \tag{8}$$

where A ,B,C,D and E stand for (i,j),$(i,j-1)$, $(i,j+1)$,$(i-1,j)$ and $(i+1,j)$, respectively.

If a random walk is then performed departing from an interior point (i,j), its trajectory will reach the boundary point p_i after a "tour" inside Ω. One must keep track of this, because many trials must be performed to have a good description of the average reward obtained at (i,j). Once a large number of random walks have reached their final values at the boundary, each boundary point p_i was the stopping place of a fraction $P_A(p_i)$ of the number of random walks generated. The approximate solution $u(A)$ is then given by

$$u(A) = g_1 P_A(p_1) + g_2 P_A(p_2) + \cdots + g_{N_{\partial\Omega}} P_A(p_{N_{\partial\Omega}}) \tag{9}$$

The analogy can be extended to more complicated equations, including variable coefficient, inhomogeneous and evolutive problems. Such discrete random walks work well although they implicitly carry the

limitation imposed by discretization errors. In principle this is a point of improvement, namely statistically solving for the local value using only the original operator without the intervenience of discretization.

Consider the presence of mixed Dirichlet-Neumann boundary conditions, that is, on a fraction of the boundary it occurs the derivative condition $\frac{\partial u}{\partial n} = 0$. To illustrate the procedure to handle this difficulty (since only Dirichlet points end random walks), consider $\hat{n} = \hat{x}$, leading to

$$u_{i,j} = \frac{1}{4} \cdot (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \tag{10}$$

$$\frac{(u_{i+1,j} - u_{i-1,j})}{2 \cdot \Delta x} = 0 \quad \textit{at a boundary point } (i,j) \tag{11}$$

where $u_{i+1,j}$ would represent a walk out of Ω. That is, to take the derivative condition into account at these boundary points the probability distribution is modified to

$$u_{i,j} = \frac{1}{4} \cdot (2 \cdot u_{i-1,j} + u_{i,j+1} + u_{i,j-1}). \tag{12}$$

A similar procedure is employed in the case of other pde's with mixed boundary conditions.

The accuracy of results obtained through the montecarlo method is highly dependent of the goodness of the random number generator (rng) employed, since at every step of every walk choices are randomly performed. Correlated trajectories are not desired, so a carefull choice of rng is necessary.

2.2.2 The algorithm ddcg

Consider the unity square Ω with boundary $\partial\Omega$. If a linear system corresponding to, say, Laplace's equation is to be solved and a five or nine-point discretization is performed, one obtains eq. (1) as the usual penta or heptadiagonal forms. The next step now would be to employ a solver such as cg, pcg, mg, or any other from such families of methods (e.g. orthomin, orthores, fas, etc.).

The strategy followed here is to postpone the construction of the linear system until a convenient domain decomposition has been performed. This is done through line drawings that generate n_{sd} balanced-size domains. A natural decomposition based on discontinuities of coefficients may occur, and this can be considered in the future.

Once each small subproblem has its boundary values specified, it may be solved in a way analogous to the global problem. To achieve this, ddcg determines the values at subdomains' boundaries through a completely parallelizable strategy, that is, montecarlo solution of the global partial differential equation in selected points of the original domain.

In all subproblems a minimization procedure as cg is now employed. After convergence is reached in all linear systems these results are brought together to generate the solution of the global problem.

Summarizing, the strategy reads

procedure domain decomposition conjugate gradient(ddcg)

1. **form Dirichlet boundary condition where Neumann b.c. provided**
2. **draw lines which form the subdomains**
3. **obtain values for interior points to define subdomains' boundaries**
4. **build local matrices**
5. **find all subdomain solutions**
6. **compose global solution**

end of ddcg

Some aspects must be mentioned. First, the macro description of ddcg above allows several combinations of numerical methods, especially for steps 3. and 5. Moreover, step 1. will not be obtained through conventional solvers, since it is of the same kind of step 3.(thus both are analyzed together). Additionally step 2. can be executed sequentially, and does not lead to heavy computation, the same being valid for step 6.

In order to take complete advantage of ddcg's structure, a parallel computational environment is mandatory, since steps 1. and 3. are performed via parallelizable monte carlo method. Moreover, steps 4. and 5. form a sequence to a subdomain. As step 4.'s termination is necessary for the solution of the smaller linear system to be started, its execution precedes that of step 5., but referring to each subproblem separately. As each subdomain refers now to an autonomous Dirichlet problem, they can all be handled simultaneously.

2.2.3 Analysis of the discrete version of ddcg

Let a partitioning procedure be implemented through multiple line drawings inside Ω. Notice that no restriction is imposed for these lines such as "they may not cross in an interior point..." or similar. In fact these lines pass by $N_{\partial\Omega_j}$ points, where the points of $\partial\Omega$ are also included. If n_v vertical and n_h horizontal lines are drawn with equal spacing, one has

$$n_{sd} = (n_v + 1) \cdot (n_h + 1) \tag{13}$$

subdomains are formed. Moreover, if n_x and n_y grid points are present in the two directions of the discrete problem, approximately

$$n_h \cdot (n_x - 2) + n_v \cdot (n_y - 2) \tag{14}$$

values have to be determined in order to start the n_{sd} local processes.

Under the simplifying assumption that it is always true that $\frac{n_h}{n_v} \approx 1$, and also $n_x \approx n_y \approx \sqrt{N_\Omega}$ one obtains

$$n_{sd} \approx (1 + n_h)^2, \tag{15}$$

and the number of interior points which form subdomain's boundaries becomes

$$N_{\partial\Omega_j} \approx 2 \cdot (n_h + 1) \cdot \sqrt{N_\Omega} \tag{16}$$

$$\approx 2 \cdot (\sqrt{n_{sd} \cdot N_\Omega}) \tag{17}$$

provided that n_{sd} is at least of order 10.

Considering the above result, one is left with the estimate

$$\frac{2 \cdot \tau \cdot \sqrt{n_{sd} \cdot N_\Omega}}{p} \tag{18}$$

for the execution of step 3. of ddcg, once p processors are available and τ is the mean time for one point's execution.

But it is well known that the discrete random walker takes on the average $\mathcal{O}(n_x^2)$ (that is $\mathcal{O}(N_\Omega)$) choices of direction before it leaves the domain. Moreover, to achieve a standard deviation of $\mathcal{O}(h^2)$, where h is the grid spacing in a uniform grid ($h = \frac{1}{N_\Omega^{1/2}}$), $\mathcal{O}(n_x^4)$ (or equivalently $\mathcal{O}(N_\Omega^2)$) trials are necessary (see Feynman, vol.I,sec.41-4). Thus one clearly obtains that τ corresponds to $\mathcal{O}(n_x^6)$ local choices of direction within random walks. That is, expected computing time in step 3. of ddcg, in units of comparisons becomes

$$\approx 2 \cdot \frac{n_{sd}^{1/2} \cdot N_\Omega^{7/2}}{p} \tag{19}$$

local "where to move this time" comparisons.

In order to handle step 5. of ddcg, one considers the usual cost estimate for cg method (cf. section 2.1). This can provide an estimate for the subproblem j of same structure as the global problem. Namely, for a certain $\epsilon_{\Omega_j} > 0$, if $p(\epsilon_{\Omega_j})$ is defined to be the smallest integer k_{Ω_j} such that

$$\|x^k_{\Omega_j} - \hat{x}_{\Omega_j}\|_{H_{\Omega_j}} \leq \epsilon \|x^0_{\Omega_j} - \hat{x}_{\Omega_j}\|_{H_{\Omega_j}} \quad , \quad \forall x^0_{\Omega_j} \in R^N_{\Omega_j} \ , \tag{20}$$

then

$$p(\epsilon_{\Omega_j}) \leq \frac{1}{2}\sqrt{K(H_{\Omega_j})} ln(\frac{2}{\epsilon_{\Omega_j}}) + 1 \tag{21}$$

where $K(H_{\Omega_j})$ is the condition number $\frac{\lambda^{\Omega_j}_N}{\lambda^{\Omega_j}_1}$ for matrix H_{Ω_j}.

In this simple model problem the ratio between the larger and the smaller eigenvalues is $\approx N_{\Omega_j}$, provided that the mesh is not strongly asymetric. Consequently one has the bound

$$p(\epsilon_{\Omega_j}) < \frac{\beta}{2}[ln(\frac{2}{\epsilon_{\Omega_j}})] \cdot N^{\frac{1}{2}}_{\Omega_j} + 1 \tag{22}$$

$$= \frac{\beta}{2}[ln(\frac{2}{\epsilon_{\Omega_j}})] \cdot \sqrt{\frac{N_\Omega}{n_{sd}}} + 1 \tag{23}$$

for each subproblem. That implies a maximum global cost roughly given by

$$n_{sd} \cdot p(\epsilon_{\Omega_j}) \leq \beta \frac{n_{sd}}{2}[\ln(\frac{2}{\epsilon_{\Omega_j}})] \cdot \sqrt{\frac{N_\Omega}{n_{sd}}}. \tag{24}$$

But each cg iteration requires one matrix-vector and a vector-vector products, with $6 \cdot N_{\Omega_j}$ floating point operations. One finally gets as bound for step 5. (in units of multiplications):

$$n_{sd} \cdot 6 \cdot N_{\Omega_j} \cdot p(\epsilon) \leq \beta \frac{\sqrt{n_{sd}}}{2}[ln(\frac{2}{\epsilon_{\Omega_j}})] \cdot \sqrt{N^3_\Omega} \ . \tag{25}$$

That is, considering the presence of p processors, and collecting these results concerning steps3. and 5. of ddcg, the total cost reads

$$\frac{2 \cdot \sqrt{n_{sd}} \cdot N^{7/2}_\Omega}{p} + \beta(\epsilon) \cdot \sqrt{n_{sd}} N^{\frac{3}{2}}_\Omega \tag{26}$$

elementary (comparison or multiplication) operations per processor.

The ddcg strategy can be conveniently implemented in any parallel configuration, but seems to be especially well suited for hybrid configurations which provide numerous weakly coupled processors to handle step3, and fewer more powerfull processors for step5. At first sight the cost seems to be prohibitive, but the improvement given by the continuous random walker will decrease such estimate.

2.2.4 Continuous random walks and ddcg

Now a different strategy is developed. It does not require carrying the truncation error of the discrete random walks. In this section the solution of a class of equations, namely the elliptic ones with constant coefficients is considered. This is an important set, since *any* elliptic partial differential equation with constant coefficients can be reduced, by suitable transformations [14], to the canonical form

$$\nabla^2 u - \lambda u = 0, \quad \lambda \ \textit{constant}. \tag{27}$$

The set considered is smaller than the one where the discrete strategy was used, but still very important. Moreover, means of representig more general elliptic operators can be investigated. The preceding equation will now be rewritten in polar coordinates, that is

$$u_{rr} + \frac{1}{r}u_r + \frac{1}{r^2}u_{\theta\theta} - \lambda u = 0. \tag{28}$$

Separating variables,

$$u(r,\theta) = R(r) \cdot \Theta(\theta), \tag{29}$$

results in two ordinary differential equations (separation constant β^2), namely

$$r^2\frac{R''(r)}{R(r)} + \frac{rR'(r)}{R(r)} - \lambda\, r^2 = \beta^2, \tag{30}$$

and

$$\Theta''(\theta) + \beta^2\Theta(\theta) = 0. \tag{31}$$

The angular equation has solutions

$$\Theta(\theta) = a_\beta cos[\beta\theta] + b_\beta sin[\beta\theta] = 0, \tag{32}$$

where a_β and b_β are independent of θ.

Requiring $\Theta(\theta)$ to be periodic in 2π, so that $u(r,\theta)$ be single valued results in

$$\beta = n = integer. \tag{33}$$

Thus the radial equation becomes

$$R''(r) + \frac{1}{r}R'(r) + (-\lambda - \frac{n^2}{r^2})\, R(r) = 0. \tag{34}$$

This is Bessel's equation, with solutions $I_n(\lambda^{1/2}r)$ and $K_n(\lambda^{1/2}r)$. The requirement that $u(r,\theta)$ be finite when r is zero discards K_n because of its singularity at zero. Thus the solution u becomes

$$u(r,\theta) = a_0\, I_0(\alpha r) + \sum_{n=1}^{\infty} I_n(\alpha r)(a_n cos[n\theta] + b_n sin[n\theta])sin[\beta\theta] = 0, \tag{35}$$

where $\alpha = \lambda^{1/2}$.

Integrating the previous equation over θ results in

$$\int_0^{2\pi} u(r,\theta)d\theta = 2\pi a_0\, I_0(\alpha r)\ . \tag{36}$$

Note that $u(0,\theta) = u(0)$ is independent of θ if u is to be single valued at r equal to zero. Setting r to zero and noting $I_0(0) = 1$ results in

$$a_0 \;=\; u_0, \tag{37}$$

that, substituted in the preceding equation yields

$$u(0) = \frac{1}{I_0(\alpha r)}\frac{1}{2\pi}\int_0^{2\pi} u(r,\theta)d\theta, \tag{38}$$

or, changing variables to $\gamma = i\alpha$ (in case $\lambda < 0$),

$$u(0) = \frac{1}{J_0(\gamma r)} \frac{1}{2\pi} \int_0^{2\pi} u(r,\theta) d\theta. \tag{39}$$

As expected, both preceding equations can be interpreted in the monte carlo framework as follows. The value of u is the average value at a circle of radius r multiplied by a factor $1/I_0(\alpha r)$, depending only on r. Thus u at the center of the circle can be computed by randomly sampling $u/I_0(\alpha r)$ on the circle. That is, a particle takes a random jump to a point $P(\theta)$ on the circle, and the statistical weight is multiplied by $w = 1/I_0(\alpha r)$. The $wu(P(\theta))$ becomes one particle's estimate of $u(0)$.

Including a more general situation in the above procedure one has: Let a particle take a random jump from the point P_0 where the value $u(P_0)$ is to be determined, to a point P_1 on the largest circle entirely contained inside Ω, with radius r_0. If P_1 lies "on" (within some small ϵ of) the boundary $\partial\Omega$, the value $u(P_1)/I_0(\alpha r_0)$ is taken as one particle's estimate of $u(P_0)$. However, in general P_1 will not lie on $\partial\Omega$, so that $u(P_1)$ has to be estimated. In this case the procedure is repeated for P_1, that is, a point P_2 is sampled on the circle of biggest radius, r_2, lying entirely inside Ω and centered on P_1. And $u(P_2)/I_0(\alpha r_1)$ is the estimate of $u(P_1)$ and hence, $u(P_2)/(I_0(\alpha r_0)I_0(\alpha r_1))$ is an estimate of $u(P_1)$.Once P_2 is on $\partial\Omega$, the one particle's estimate is obtained, otherwise $u(P_2)$ will be determined in the same fashion.

Finally, each trajectory generated by the "walk on biggest circles" described above will give a sequence of points $P_1, \cdots, P_n$ (P_n lies on the boundary). Then the corresponding one trajectory's estimate will be given by

$$\frac{u(P_n)}{I_0(\alpha r_0) \cdots I_0(\alpha r_n)}. \tag{40}$$

After enough random trajectories are generated, the average of the "one trajectory" estimates will give a good description of $u(P_0)$. Some numeric data concerning the continuous version are presented in this text, and encourages further developments.

3 Codes and Numerical Tests

3.1 Code for the Model Problem

A description of the code is presented, and implementation choices are addressed.

```
program ddcg
   init_routine
   for node at subdomain boundaries do
         mcar_routine (node,aux)
         value ← aux
   end_for
   for all domains (i) do
         solve_routine (i)
         place_values (i)
   end_for
    fprints
end_ddcg
```

At init, the algorithm performs the subdivision of Ω in domain subregions. Next, the grid points on those lines at the boundary of subdomains have their values determined through mcar_routine, where continuous (or discrete) random walks are employed, with the aid of a random number generator (IMSL's ggubs). This choice is motivated by the efficiency and long period of the pseudorandom numbers offered by such routine, together with its uniform distribution of values.

Following ddcg, solve_routine (in this implementation cg method is employed, and a Gauss-Seidel version is also available) is performed to all subdomains, and the values thus obtained have to be placed correctly at the original problem. The step place_values stands for such part, in which one bus is employed and frees the corresponding processor for other tasks.

3.2 Numerical Tests

In this section the effectiveness and efficiency of ddcg are addressed. The computational environment is a sequential one, so a normalization will be employed when required.

The first example chosen is the simple equation ∞, or simply

$$\nabla \cdot [\vec{u}C + D\nabla C] = 0\,. \tag{41}$$

Consider $\vec{u}$ and D constants. Boundary conditions can be chosen that determine the solution

$$C = e^{-x} \cdot e^{-y}\,, \tag{42}$$

once $2D = u_x + u_y$ and $D = 1 = u_x = u_y$.

In the following tables resulting values for selected points in Ω are presented as a function of the number of trials, by means of the application of the continuous discrete-version of DDCG. Such points are used for they represent the alternatives of distances to the boundary. As a rule, the generation of random number sequences could be considered apart from the computational cost, since millions of numbers can be stored to avoid spending this computing time during execution of the algorithm. In all simulations reported, however, the generation of random numbers is performed within the computation.

As a standard the values for the bi-dimensional example are:

$ntrials$	1000
$n_x = n_y$	65
eps_{mc}	$0.1 * \Delta x = 0.0015625$
$epsgc$	0.001
$n_{subdomains}$	4

In table 1 some results are listed, which illustrate the behavior of the ddcg solver, showing the final values obtained after a complete ddcg cycle at selected points.

$P_1 = (\frac{1}{2}, \frac{1}{2})$	*ntrials*	Result	Cost(s)
	$1.0 * 10^2$	0.33932	40.2
	$5.0 * 10^2$	0.36198	93.4
	$1.0 * 10^3$	0.36684	159.5
	$5.0 * 10^3$	0.36757	686.9
	$1.0 * 10^4$	0.36909	1365.3
	∞ (exact)	0.36788	
$P_2 = (1 - \frac{1}{64}, \frac{1}{2})$	$1.0 * 10^2$	0.26482	40.2
	$5.0 * 10^2$	0.26492	93.4
	$1.0 * 10^3$	0.26505	159.5
	$5.0 * 10^3$	0.26494	686.9
	$1.0 * 10^4$	0.26498	1365.3
	∞ (exact)	0.26497	
$P_3 = (1 - \frac{1}{64}, 1 - \frac{1}{64})$	$1.0 * 10^2$	0.16314	40.2
	$5.0 * 10^2$	0.16328	93.4
	$1.0 * 10^3$	0.16332	159.5
	$5.0 * 10^3$	0.16325	686.9
	$1.0 * 10^4$	0.16325	1365.3
	∞ (exact)	0.16325	

Table 1 - Numerical tests of effectiveness of ddcg
as a function of number of random walks for
$N_\Omega = 4225$ and $n_{sd} = 4$

Next, the whole procedure ddcg is tested for selected points in a similar fashion, expressing results as a function of the number of subdomains. The best combination of number of random walks for each case (according to table 1) is adopted. Such results are shown in table 2. Namely, P_1 is chosen near the boundary of Ω in both dimensions, P_2 near Ω in only one direction and near the center of the other coordinate, and P_3 is at the center of Ω. In all these tests the required precision in the conjugate gradient procedure is 10^{-5} (giving an averaged maximum of 10^{-4} per component of the gradient).

	n_{sd}	Result	Cost(s)
$P_1 = (\frac{1}{2}, \frac{1}{2})$	1	0.36788	40.2
	2	0.36684	104.0
	4	0.36684	159.9
	8	0.36684	285.7
	16	0.36684	406.4
	32	0.36684	654.3
	64	0.36684	882.4
	∞(exact)	0.36788	
$P_2 = (1 - \frac{1}{64}, \frac{1}{2})$	1	0.26497	40.2
	2	0.26499	104.0
	4	0.26505	159.9
	8	0.26487	285.7
	16	0.26497	406.4
	32	0.26524	654.3
	64	0.26497	882.4
	∞(exact)	0.26497	
$P_3 = (1 - \frac{1}{64}, 1 - \frac{1}{64})$	1	0.16325	40.2
	2	0.16331	104.0
	4	0.16332	159.9
	8	0.16330	285.7
	16	0.16337	406.4
	32	0.16324	654.3
	64	0.16328	882.4
	∞(exact)	0.16325	

Table 2 - Numerical tests of effectiveness for ddcg
as a function of number of subdomains for
$N_\Omega = 4225$ and $ntrials = 10^3$

Now a convenient scaling is performed, in order to allow a comparison with other algorithms. By virtue of the structure of ddcg, message passings are neglected at this point. On true parallel environments one will measure how accurate this approximation (ddcg's speedup= 1) is.

number_of_processors	cost(s)
1	49.4
2	51.99
4	39.98
8	35.7
16	25.40
32	20.45
64	13.79

Consider now the discrete case, for which the same equation is chosen, but with boundary conditions that correspond to the solution $x^2 - y^2$. Moreover, the number of grid points is 1089, 4 subdomains are used, with *eps* within conjugate gradient being 10^3. Some typical results read as in table 3:

$P_1 = (\frac{1}{2}, \frac{1}{2})$	*ntrials*	Result	Cost(s)
	$1.0 * 10^2$	0.01204	52.0
	$5.0 * 10^2$	−0.01673	214.8
	$1.0 * 10^3$	0.00102	414.2
	$5.0 * 10^3$	0.00452	2075.3
	∞ (exact)	0.00000	
$P_2 = (1 - \frac{1}{32}, \frac{1}{2})$	$1.0 * 10^2$	0.65710	52.0
	$5.0 * 10^2$	0.69311	214.8
	$1.0 * 10^3$	0.68972	414.2
	$5.0 * 10^3$	0.68757	2075.3
	∞ (exact)	0.68848	
$P_3 = (1 - \frac{1}{32}, 1 - \frac{1}{32})$	$1.0 * 10^2$	0.00009	52.0
	$5.0 * 10^2$	−0.00006	214.8
	$1.0 * 10^3$	0.00006	414.2
	$5.0 * 10^3$	0.00003	2075.3
	∞ (exact)	0.00000	

Table 3 - Numerical tests of effectiveness for the discrete version of ddcg

After one notes the innadequacy of the discrete scheme, consider again the continuous case, but on a three dimensional domain. The results reported here refer to a 13^3 grid, 8 subdomains, $eps = 10^3$ within the conjugate gradient method, and $eps_mc = 0.1 * \Delta x = 0.00833$ for the montecarlo step. The model equation is the same adopted for the continuous 2-dimensional problem above, thus corresponding to the solution $e^{-x} \cdot e^{-y} \cdot e^{-z}$. These results are summarized in table 4, that is,

$P_1 = (\frac{1}{2}, \frac{1}{2})$	*ntrials*	Result	Cost(s)
	$1.0 * 10^2$	0.06375	61.8
	$5.0 * 10^2$	0.06391	256.2
	$1.0 * 10^3$	0.06389	500.1
	$4.0 * 10^3$	0.06392	1952.7
	∞ (exact)	0.06393	
$P_2 = (1 - \frac{1}{12}, \frac{1}{2}, 1 - \frac{1}{12})$	$1.0 * 10^2$	0.10515	61.8
	$5.0 * 10^2$	0.10550	256.2
	$1.0 * 10^3$	0.10487	500.1
	$4.0 * 10^3$	0.10570	1952.7
	∞ (exact)	0.10540	
$P_3 = (\frac{1}{2}, \frac{1}{2}, 1 - \frac{1}{12})$	$1.0 * 10^2$	0.24086	61.8
	$5.0 * 10^2$	0.22429	256.2
	$1.0 * 10^3$	0.21913	500.1
	$4.0 * 10^3$	0.22263	1952.7
	∞ (exact)	0.22313	

Table 4 - Numerical tests of effectiveness for 3-dimensional ddcg

4 Concluding Remarks

4.1 Sources of Errors

The limitation of working out the discrete procedure after a discretization has taken place leads to the usual truncation error. In the example presented, the five point formula adopted carries an error of $\mathcal{O}(h^2)$. To overcome this a mesh refinement would be required, followed by a restart, at an excessive cost. The truncation error does not exist in the continuous case.

Some variation reduction techniques can also be tried in order to make the local solution of the problem in interior points cheaper. The usual bound for $1-d$ discrete random walks, which is the basis of the whole analysis, is too poor to allow an implementation competitive with conventional solvers.

Another kind of error (sampling) is due to the fact that a statistical strategy is performed at several internal points. But the correct value would appear after infinite samples have been used. A feasable computing time, together with a required accuracy can delimit this part, as a real time decision.

Use of a poor random number generator can cause correlation between different trajectories (random walks), thus affecting local results. In the results reported, however, this was not observed.

4.2 Efficiency and effectiveness

As evidenced in the numerical results, the efficiency of the procedure is not high for the discrete case. In the continuous one there is an improvement, since the dependence of the number of random jumps is smoother as a function of the total number of grid points in such case than it is for the discrete one.

The effectiveness of the procedure is clear from numeric data as well as from the theoretical analysis, in both discrete and continuous situations. The discrete version does not benefit from the exact local calculation, due to the truncation error. In practice, only the continuous implementations are recomended.

A remarkable property is the complete parallelizability of ddcg, in particular when a loosely coupled environment is considered. This case is particularly interesting, since the cost involved in the development of the hardware is orders of magnitude lower than the strongly coupled "supercomputer" environments. Moreover, the advent of applicative softwares for such cheaper environments leads to smaller global cost of computations, even when computing time is slightly higher.

A very important aspect is that the average number of jumps depends only (for a given ϵ) on the dimensionality of the domain, in the continuous method. Thus the increase is from 2 (exact) in 1-dimensional to 5 in the 2-dimensional case and to 11 in the 3-d one. That is, the number of grid points affects the total computing time as $\mathcal{O}(N^1_{\partial\Omega_j})$. But at this point one recalls that the number of internal boundaries' grid points are the defining parameters, together with monte carlo's ϵ. Moreover, as previously described, the number of points to calculate through monte carlo depend on the total number of grid points as $\mathcal{O}(N^1_\Omega)$. But this means that, what one in fact has is a means with cost $\mathcal{O}(N^1_\Omega)$ that decomposes the domain and prepares n_{sd} subproblems that do not depend on each other. That is, consider using a spectral or multigrid method for solving all subproblems. In such case **the classs of methods reported here shows an asymptotic rate of convergence competitive with multigrid methods, with complete** (neglecting message passings) **parallelizability in loosely coupled parallel** (not massively,i.e., number of processors $\mathcal{O}(10^3)$) **environments.**

The extension of the "walk on biggest circles" concept to non constant coefficients and to equations with a non constant source term, such as Poisson's equation [11], are under investigation, but does not affect the favorable properties obtained.

References

1- Albrecht,P. ; "Análise Numérica- Um Curso Moderno" - ed.LTC (1973)

2- Ashby,S. - "A taxonomy for conjugate gradient methods"; SIAM J. Numer. Anal., **27** pp.1542-1568 (1990)

3- Ashby,S. - "Polynomial preconditioning"; Research Report (1991)

4- Axelsson,O & Barker,V.A. - "Finite Element Solution of Boundary Value Problems - Theory and Computation";ed. Academic Press (1984)

5- Axelsson,O ; "Preconditioned conjugate gradient methods"; BIT no. (1989)

6- Booth, T.E. ; "Exact monte carlo solution of elliptic partial differential equations"; J. Comp. Phys. **39**, 396-404 (1981)

7- Booth, T.E. ; "Regional monte carlo solution of elliptic partial differential equations"; J. Comp. Phys. **47**, 281-290 (1981)

8- Bramble,J.H.,Pasciak,J.E.,Schatz,A.H.; "The construction of preconditioners for elliptic problems by substructuring"; I. [vol. 47,No 175, 103-134]; II. [vol 49, No.170, 1-16]-Math.Comp. (1986)

9- Bramble,J.H., Pasciak,J.E., Xu,J ; "Parallel multilevel preconditioners"; Math. Comp. vol 55 no 191 (1990)

10- Brandt,A.;"Multilevel adaptive solutions to boundary-value problems" - Mathematics of Computation **31**,333-390 (1977)

11- DeLaurentis, J. M. & Romero, L. A.; "A Monte Carlo Method for Poisson's Equation"; J. Comp. Phys. **90**, 123-140 (1990)

12- Farlow,S.; "Partial Differential Equations - for scientists and engineers"; John Wiley & Sons, Inc.; (1982)

13- Fedorenko,R.P.; URSS-Computational and Mathematical Physics 4 227 (1964)

14- Garbedian, P.R. ; "Partial Differential Equations", ed. Wiley, New York, (1964)

15- Hackbush,W. & Trottenberg,U.,eds.; "Multigrid Methods"- Lecture Notes in Mathematics No. 960- Springer, Berlin (1982)

16- Hammersley,J.M. & Handscomb,D.C. ; "Monte Carlo Methods"; ed. Methuen (1964)

17- Hestenes & Stiefel; "Methods of conjugate gradients for the solution of linear systems of equations"; J. Res. National Bureau of Standards 49, 409-436 (1952)