Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2012, Article ID 894074, 15 pages
doi:10.1155/2012/894074

Research Article

A Parallel Wavelet-Based Algebraic Multigrid Black-Box Solver and Preconditioner

Fabio Henrique Pereira[1] and Sílvio Ikuyo Nabeta[2]

[1] *Industrial Engineering Post Graduation Program, Nove de Julho University (PMEP/UNINOVE), Francisco Matarazzo Avenue, 612, 05001100 São Paulo, SP, Brazil*
[2] *Electrical Machine and Drives Lab, São Paulo University (GMAcq/EP/USP), Luciano Gualberto Avenue, 380, 05508-010 São Paulo, SP, Brazil*

Correspondence should be addressed to Fabio Henrique Pereira, fabiohp@uninove.br

Received 1 November 2011; Revised 26 January 2012; Accepted 8 February 2012

Academic Editor: Massimiliano Ferronato

Copyright © 2012 F. H. Pereira and S. I. Nabeta. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This work introduces a new parallel wavelet-based algorithm for algebraic multigrid method (PWAMG) using a variation of the standard parallel implementation of discrete wavelet transforms. This new approach eliminates the grid coarsening process in traditional algebraic multigrid setup phase simplifying its implementation on distributed memory machines. The PWAMG method is used as a parallel black-box solver and as a preconditioner in some linear equations systems resulting from circuit simulations and 3D finite elements electromagnetic problems. The numerical results evaluate the efficiency of the new approach as a standalone solver and as preconditioner for the biconjugate gradient stabilized iterative method.

1. Introduction

The algebraic multigrid (AMG) method is one of the most efficient algorithms for solving large sparse linear systems. Especially in the context of large-scale problems and massively parallel computing, the most desirable property of AMG is its potential for algorithmic scalability: in the ideal case, for a matrix problem with n unknowns, the number of iterative V-cycles required for convergence is independent of the problem size n and the work in the setup phase and in each V-cycle is linearly proportional to the problem size n [1, 2]. For all this, the need to solve linear systems arising from problems posed on extremely large, unstructured grids has been generating great interest in parallelizing AMG.

However, there are two major problems: first, the core of the AMG setup phase includes the grid coarsening process, which is inherently sequential in nature [1–3]. This coarsening scheme, for traditional AMG, can lead to computational complexity growth as the problem size increases, resulting in an elevated memory use and execution time and in a reduced scalability [4, 5]. Second, most parallel AMG algorithms are based on domain decomposition ideas, which have been proved to be very efficient but require a hierarchy of meshes that eliminates the algebraic characteristic of AMG and precludes its use as a black-box method.

Due to those difficulties and the importance of the development of efficient parallel preconditioners for large, sparse systems of linear equations, the investigation of new parallel approaches has been the main subject of many researchers [6–12]. In this context, a great amount of work has been aimed to extract some parallelism from serial preconditioners such as factorization-based methods, which provide effective preconditioning on sequential architectures [8–10]. However, scalable parallel implementation of incomplete factorization preconditioners presents many limitations and challenges, and although some interesting approaches have presented good performance for certain classes of problems, quite scalable parallel algorithms for this kind of preconditioners seem to have not been available [8].

Also has received much attention in the last years the development of preconditioning approaches that have inherently parallel characteristics. In particular, approximate inverse preconditioners have proven extremely promising and effective for the solution of general sparse linear systems of equations [6, 7, 10–12]. Unfortunately, this kind of preconditioner also has some drawbacks: in general, as the discretization is refined the amount of work per grid point grows with problem size, and it is inherently difficult to approximate the inverse of very ill-conditioned linear systems with a sparse matrix [13].

In this work we introduce a new parallel algorithm for wavelet-based AMG (PWAMG) using a variation of the parallel implementation of discrete wavelet transforms. This new approach eliminates the grid coarsening process present at standard AMG setup phase, simplifying significantly the implementation on distributed memory machines and allowing the use of PWAMG as a parallel black-box solver and preconditioner. The parallel algorithm uses the message passing interface (MPI) that provides a standard for message passing for parallel computers and workstation clusters.

A sequential version of WAMG was introduced recently [14], and it has revealed to be a very efficient and promising method for several problems related to the computation of electromagnetic fields [15, 16]. Here, the method is used as a parallel black-box solver and as a preconditioner in some linear equations systems resulting from circuit simulations and 3D finite elements electromagnetic problems. The numerical results evaluate the efficiency of the new approach as a standalone solver and as preconditioner for the biconjugate gradient stabilized iterative method.

2. The Discrete Wavelet Transform

The discrete wavelet transform (DWT) corresponds to the application of low-pass and high-pass filters, followed by the elimination of one out of two samples (decimation or subsampling). The discrete signal, which in one dimension is represented by a vector of values, is filtered by a set of digital filters that are associated to the wavelet adopted in the analysis.

Starting from a vector $y(N)$ at level 0, two sets of coefficients are generated in each level l of the process: a set d_l of wavelets coefficients (detail coefficients) and a set c_l of approximation coefficients. This procedure can be applied again, now using c_l as an input vector to create new coefficients c_{l+1} and d_{l+1}, successively.

Very important classes of filters are those of finite impulse response (FIR). The main characteristic of these filters is the convenient time-localization properties. These filters are originated from compact support wavelets, and they are calculated analytically. An example of FIR filters is the length-2 scaling filter with Haar or Daubechies-2 coefficients, which are given by (2.1):

$$h_{D2} = [h_0, h_1] = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]. \tag{2.1}$$

For more details about compact FIR filters see [17].

For a given vector $y = (y_1, y_2, \ldots, y_N)$, the Haar wavelet transform creates an approximation vector c and a detail vector d according to (2.2) and (2.3), respectively:

$$c = (c_1, c_2, \ldots, c_{N/2}), \quad \text{with } c_i = \frac{(y_{2i-1} + y_{2i})}{\sqrt{2}}, \tag{2.2}$$

$$d = (d_1, d_2, \ldots, d_{N/2}), \quad \text{with } d_i = \frac{(y_{2i-1} - y_{2i})}{\sqrt{2}}. \tag{2.3}$$

It is not difficult to see that this procedure will work only if N is even.

In the 2D case, in which the discrete signal is represented by a matrix, the DWT is obtained through the application of successive steps of 1D transform into the rows and columns of the matrix. This process generates a matrix formed by four types of coefficients: the approximation coefficients and the detail coefficients (horizontal, vertical, and diagonal), as illustrated in Figure 1. Blocks H and G represent, respectively, the low-pass and high-pass filters.

In both cases, the approximation coefficients keep the most important information of the discrete signal, whereas the detail coefficients possess very small values, next to zero. These approximation coefficients will contain low-pass information, which is essentially a low-resolution version of the signal and represent a coarse version of the original data.

3. A Wavelet-Based Algebraic Multigrid

The approximation property of wavelet is explored by wavelet-based algebraic multigrid for creating the hierarchy of matrices. The method considers the use of a modified discrete wavelet transform in the construction of the transfer operators and the hierarchy of matrices in the multigrid approach.

A two-dimensional modified DWT is applied to produce an approximation of the matrix in each level of the wavelets multiresolution decomposition process. An operator formed only by low-pass filters is created, which is applied to the rows and columns of the matrix. This same operator is used for the intergrid transfer in the AMG. If a length-2 (first order) scaling filter is used, for example, which means the application of the operation

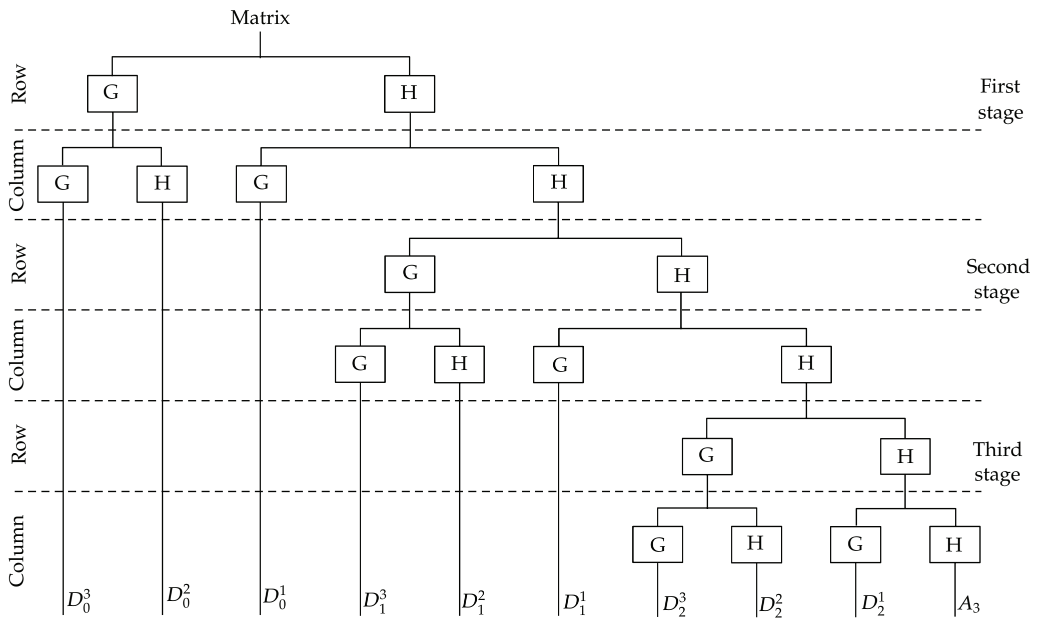


Figure 1: The two-dimensional DWT: one-dimensional transform in the rows and columns of the matrix.

defined in (2.2) in the rows and columns of the matrix, the operation is matricially defined as (3.1):

$$P_k^{k+1} = \begin{pmatrix} h_1 & h_0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & h_1 & h_0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & \vdots & & & & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_0 \end{pmatrix}_{(N/2)\times N}. \tag{3.1}$$

The prolongation operator is defined in the usual form,

$$P_{k+1}^k = \left(P_k^{k+1}\right)^T, \tag{3.2}$$

and the matrix in the corresponding level k with the Galerkin condition:

$$A^{k+1} = P_k^{k+1} A^k P_{k+1}^k, \tag{3.3}$$

reminding that $A^0 = A$ is the matrix of the original system.

Once the intergrid transfer operators and the hierarchy of matrices are created, the multigrid method (V-cycle) can be defined as usual, using a recursive call of the following two-level algorithm.

Algorithm 3.1 (Two-level multigrid). The following holds.

```
Input: the right hand side vector $b$, the original matrix $A$
        and the coarse grid matrix $\mathrm{PAP}^T$
Output: approximation $\tilde{x}$

  (1) Choose an initial guess $x$ and apply $\nu_1$ smoothing steps in $Ax = b$
  (2) Compute $r = b - Ax$
  (3) $e = (\mathrm{PAP}^T)^{-1} Pr$
  (4) $\tilde{x} = x + P^{\mathrm{T}} e$
  (5) Apply $\nu_2$ smoothing steps in $A\tilde{x} = b$
  (6) Return $\tilde{x}$.
```

In this paper, a V-cycle multigrid with $\nu_1 = \nu_2 = 1$ is applied iteratively as a solver and also as a preconditioner inside the biconjugate gradient stabilized (BiCGStab) iterations.

3.1. The Choice of the Wavelet Filters

A common problem on the choice of the filters is to decide between the fill-in control and the wavelet properties. As the WAMG often deals with sparse matrices, the control of the nonzero number is a very important task. In this case, if the matrix A^k is sparse, then the number of nonzero elements in the next level matrix $A^{k+1} = P_k^{k+1} A^k P_{k+1}^k$ will depend on the order of the filter used in the restriction and prolongation operators. In fact, the longer the filter used the larger the number of nonzero entries in the next computed matrix. Consequently, most of the wavelet-based multigrid methods use shorter filters such as Haar or Daubechies-2 coefficients in its approaches [18–23]. This is also the case in this paper.

4. A Parallel Wavelet-Based Algebraic Multigrid

One advantage of the proposed approach is the elimination of the coarsening scheme from the setup phase. The application of the operations defined in (2.3) in the rows and columns of a matrix allows creating an approximation without any information about meshes or the adjacency graph of the matrix. Thus the implementation on distributed memory machines becomes simpler allowing the use of the method as a parallel black-box solver and preconditioner. Our approach, based on the parallel implementation of discrete wavelet transform presented in [24], avoids any communication among processors in the setup phase if first-order filters are used.

The parallelization strategy starts dividing equally the number N of rows of the matrix between the processors, in such a way that the 1-D transform defined in (2.3) could be applied entirely locally in the rows. As each matrix column is equally divided, the transformation in this direction is accomplished partially in each processing unit, considering the size of the column in its local memory. For np processors, for example, each processor will apply the wavelet transform in the $\lceil N/np \rceil$ elements of the column in its memory, where $\lceil x \rceil$ means the largest integer even less than x. If the largest integer less than N/np is odd, then the last element of the column in the local memory is unchanged, as illustrated in Figure 2 for $N = 10$ and $np = 2$. In the figure, $c_{i,j}^l$ means the ith local element of the level l vector c in the processor j.

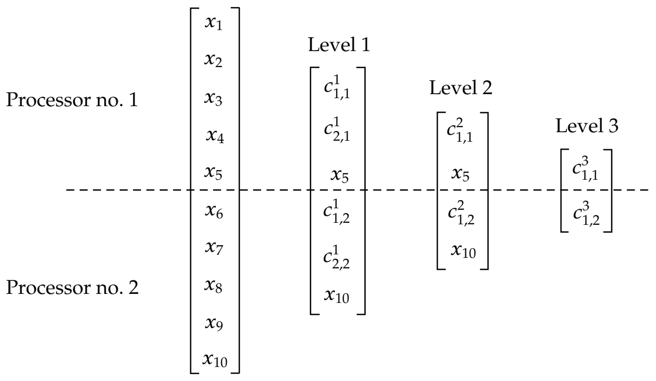

Processor no. 1
Processor no. 2
Level 1
Level 2
Level 3
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
$c^1_{1,1}$ $c^1_{2,1}$ x_5 $c^1_{1,2}$ $c^1_{2,2}$ x_{10}
$c^2_{1,1}$ x_5 $c^2_{1,2}$ x_{10}
$c^3_{1,1}$ $c^3_{1,2}$

Figure 2: Illustration of the 1D parallel wavelet transform with 2 processors.

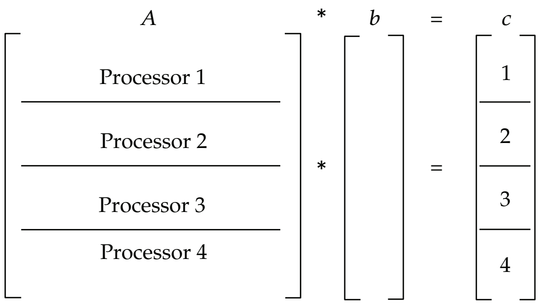


Figure 3: Matrix-vector product of a matrix distributed by rows.

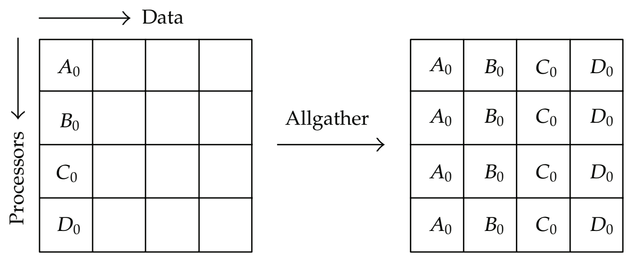


Figure 4: MPI collective communication function.

Thus the resulting coarse level matrix is calculated entirely locally.

In the solver phase the interprocessors communication is necessary only for operations involving matrices. More specifically, it is necessary to update the vector always after the smoothing step and after the matrix-vector product in the residual calculation (lines 1, 2, and 5 of the algorithm). In a matrix-vector product $A * b = c$, for example, the matrix A is distributed in rows, the vector b is shared by all processors, and vector c is calculated in parallel as illustrated in Figure 3, for 4 processors. Then the resulting vector c is updated by the processors through the message passing interface library (MPI). This task is accomplished by using the MPI collective communication function `MPI_Allgather` [25]. The `MPI_Allgather` function effect is shown in Figure 4. It is important to highlight that only the resulting vector should be updated. It means each processor communicates to the other only a few elements

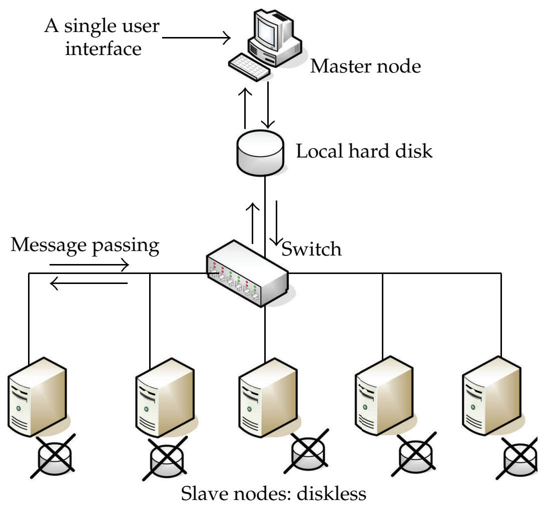


Figure 5: The Beowulf Linux cluster architecture.

Table 1: Electromagnetic matrices properties.

Matrix properties	*2cubes_sphere*	*Offshore*	*dielFilterV2real*	*Circuit5M_dc*
Number of rows	101,492	259,789	1,157,456	3,523,317
Number of columns	101,492	259,789	1,157,456	3,523,317
Nonzeros	1,647,264	4,242,673	48,538,952	14,865,409
Explicit zero entries	0	0	0	4,328,784
Type	Real	Real	Real	Real
Structure	Symmetric	Symmetric	Symmetric	Unsymmetric
Positive definite?	Yes	Yes	No	No

Table 2: Results for sequential tests (not enough memory).

Matrix	ILU + BiCGStab			WAMG			WAMG + BiCGStab		
	Setup	Solver	n	Setup	Solver	n	Setup	Solver	n
Circuit5M_dc	6.96	29.98	3	47.4	14.37	3	61.30	45.49	2
2cubes_sphere	1.98	2.15	3	5.29	2.02	3	5.30	1.55	4
Offshore	5.36	21.55	10	16.67	18.42	6	16.69	21.17	10
dielFilter-V2real	—	—	—	—	—	—	—	—	—

that it stores. However, in order for the process to continue, the whole vector must be updated on each processor and some kind of collective communication should take place.

The PWAMG method in this work uses a hybrid Jacobi-Gauss method as smoother and the V-cycle for the resolution scheme. The Gauss-Seidel smoothing is applied inside each processor and the Jacobi method applied interprocessors.

5. The Numerical Test Problems

The parallel algorithm uses the version one of the MPI that provides a standard for message passing for parallel computers and workstation clusters. The method has been implemented

Table 3: Parallel results for *2cubes_sphere* matrix.

	np	Nrows	Nonzeros	PWAMG solver Setup time CPU (Wtime)	PWAMG solver Solver time CPU (Wtime)	n	PWAMG BiCGStab Setup time CPU (Wtime)	PWAMG BiCGStab Solver time CPU (Wtime)	n
2cubes_sphere	1	101492	1647264	5.29	2.02	3	5.30	1.55	4
	2	50746	828332	2.43 (2.43)	1.84 (2.99)	3	2.40 (2.40)	6.81 (9.83)	7
		50746	818932	2.20 (2.20)	2.22 (2.99)		2.17 (2.17)	5.74 (9.83)	
	3	33830	552260	1.61 (1.62)	1.41 (3.46)	3	1.62 (1.62)	5.50 (12.09)	7
		33830	550206	1.50 (1.50)	1.15 (3.47)		1.49 (1.49)	3.63 (12.09)	
		33832	544798	1.42 (1.42)	1.74 (3.47)		1.41 (1.41)	4.36 (12.08)	
	4	25373	415150	1.19 (1.20)	1.12 (3.26)	3	1.19 (1.19)	4.18 (11.48)	7
		25373	412416	1.12 (1.12)	0.96 (3.22)		1.12 (1.12)	3.83 (11.48)	
		25373	406516	1.07 (1.07)	1.01 (3.23)		1.06 (1.06)	3.27 (11.46)	
		25373	413182	1.06 (1.06)	1.18 (3.26)		1.06 (1.06)	3.22 (11.46)	
	5	20298	333205	0.95 (0.95)	1.00 (3.29)	3	0.96 (0.94)	4.27 (11.87)	7
		20300	330383	0.88 (0.88)	0.77 (3.29)		0.88 (0.88)	3.35 (11.85)	
		20298	330868	0.88 (0.88)	0.78 (3.30)		0.87 (0.87)	2.70 (11.85)	
		20298	328976	0.84 (0.83)	0.75 (3.31)		0.85 (0.84)	2.68 (11.87)	
		20298	323832	0.85 (0.85)	1.08 (3.32)		0.84 (0.84)	2.59 (11.85)	
	6	16915	277472	0.81 (0.82)	0.87 (3.27)	3	0.81 (0.81)	4.28 (11.95)	7
		16915	276059	0.74 (0.74)	0.69 (3.24)		0.74 (0.74)	3.15 (11.94)	
		16915	274147	0.73 (0.74)	0.71 (3.24)		0.74 (0.74)	2.40 (11.93)	
		16915	269896	0.73 (0.73)	0.72 (3.24)		0.74 (0.74)	2.32 (11.94)	
		16915	274788	0.69 (0.68)	0.68 (3.24)		0.72 (0.72)	2.32 (11.94)	
		16917	274902	0.72 (0.72)	1.03 (3.26)		0.68 (0.68)	2.32 (11.93)	

using C++ and tested in a homogeneous Beowulf cluster with 6 machine nodes (Core 2 Duo, 1 GB RAM) connected to a switched fast Ethernet network, as illustrated in Figure 5.

The multigrid V-cycle approach was applied as a resolution scheme to solve some linear equations systems resulting from circuit simulations and finite elements electromagnetic problems. The parallel method was also applied as a preconditioner for the iterative biconjugate gradient stabilized (BiCGStab) method, which has been implemented using the same vector updating approach.

The three first matrices are related to 3D finite element electromagnetic analysis. The matrices *2cubes_sphere* and *offshore* are from a study of edge-based finite-element time domain solvers for 3D electromagnetic diffusion equations. The *dielFilterV2real* is a real symmetric matrix, which comes from a high-order vector finite element analysis of a 4th-pole dielectric resonator [26]. The last real unsymmetric matrix is from circuit simulation problems from Tim Davis sparse matrix collection [27]. The matrices properties are presented in Table 1.

6. Results

The problems were solved firstly using the sequential version of the proposed method. For comparison, the BiCGStab method preconditioned by the incomplete LU was used.

Table 4: Parallel results for *offshore* matrix.

	np	Nrows	Nonzeros	PWAMG solver			PWAMG BiCGStab		
				Setup time CPU (Wtime)	Solver time CPU (Wtime)	n	Setup time CPU (Wtime)	Solver time CPU (Wtime)	n
Offshore	1	259789	4242673	16.67	18.42	6	16.69	21.17	10
	2	129894	2120163	8.13 (8.12)	4.65 (7.71)	3	8.08 (8.08)	31.94 (46.27)	12
		129895	2122510	7.38 (7.38)	5.80 (7.71)		7.37 (7.37)	26.93 (46.28)	
	3	86596	1424747	5.13 (5.15)	3.46 (9.45)	3	5.07 (5.07)	31.0 (64.24)	14
		86596	1410644	4.93 (4.93)	3.25 (9.50)		4.97 (4.97)	22.6 (64.19)	
		86597	1407282	5.01 (5.00)	4.97 (9.50)		4.90 (4.90)	20.6 (64.24)	
	4	64947	1065971	3.60 (3.61)	2.96 (8.74)	3	4.13 (4.12)	19.47 (60.21)	14
		64947	1060994	4.15 (4.14)	2.67 (8.64)		3.74 (3.73)	19.31 (60.18)	
		64947	1059169	3.42 (3.42)	2.73 (8.68)		3.54 (3.55)	16.98 (60.09)	
		64948	1056539	3.80 (3.80)	3.19 (8.74)		3.39 (3.38)	16.53 (60.13)	
	5	51957	858175	2.77 (2.76)	2.50 (8.76)	3	3.28 (3.28)	21.64 (60.97)	14
		51957	847435	3.31 (3.30)	2.31 (8.74)		2.94 (2.94)	16.83 (60.94)	
		51957	846747	2.96 (2.95)	2.23 (8.77)		2.88 (2.87)	14.76 (60.90)	
		51957	845466	2.87 (2.87)	1.99 (8.80)		2.86 (2.85)	14.28 (60.94)	
		51961	844850	2.93 (2.93)	3.13 (8.81)		2.75 (2.75)	13.83 (60.97)	
	6	43298	715228	2.22 (2.21)	2.21 (8.87)	3	2.61 (2.61)	22.69 (61.66)	13
		43298	709519	2.62 (2.61)	2.07 (8.90)		2.60 (2.60)	14.01 (61.60)	
		43298	707023	2.62 (2.61)	2.09 (8.90)		2.52 (2.52)	13.63 (61.65)	
		43298	705998	2.20 (2.20)	1.87 (8.89)		2.51 (2.51)	13.59 (61.63)	
		43298	703621	2.54 (2.53)	1.98 (8.90)		2.20 (2.20)	12.52 (61.66)	
		43299	701284	2.29 (2.29)	3.09 (8.87)		2.18 (2.19)	12.29 (61.63)	

The results are presented in Table 2. In all cases, the convergence is defined by $\|r^n\|/\|b\| < 10^{-6}$, where r^n is the residual vector at the nth iteration and the right hand side vector b is chosen so that the solution is a unitary vector. The setup and solver times are in seconds, and n is the number of iterations.

The parallel results for the circuit simulation and electromagnetics problems are presented in Tables 3, 4, 5, and 6 for the PWAMG solver and preconditioner. The setup and solver times are in seconds and have been presented in the form $t1$ ($t2$), where $t1$ is the processing time returned by the C clock() function and $t2$ is the total time spent in the corresponding phase, which includes the MPI communication time and is measured using the MPI function MPI_Wtime().

In some cases the CPU and wall-clock times are nearly equal, which means that the processes that are waiting for synchronization are still consuming full CPU time. Moreover, the CPU time and the wall-clock time are practically the same in setup phase indicating that there are no interprocessor communications during the setup. It is a great advantage of the proposed approach since this phase is the most time-consuming task in multilevel approaches, as can be seen in Table 2. For single processor algorithms the times $t1$ and $t2$ are the same.

Table 5: Parallel results for *circuit5M_dc* matrix.

	np	*N*rows	Nonzeros	PWAMG solver			PWAMG BiCGStab		
				Setup time CPU (Wtime)	Solver time CPU (Wtime)	*n*	Setup time CPU (Wtime)	Solver time CPU (Wtime)	*n*
	1	3523317	14865409	47.4	14.37	3	61.30	45.49	2
	2	1761659	8279908	27.73 (27.89)	5.86 (21.51)	3	27.76 (27.90)	14.37 (37.37)	2
		1761658	6585501	11.94 (11.94)	10.56 (21.24)		11.96 (11.94)	6.63 (40.58)	
		1174439	5531627	19.83 (19.84)	4.88 (27.07)		19.83 (19.82)	5.24 (27.43)	
	3	1174439	4956074	10.95 (10.95)	3.79 (27.05)	3	10.96 (10.95)	4.8 (26.59)	2
		1174439	4377708	8.03 (8.03)	4.07 (26.25)		8.03 (8.02)	4.1 (27.39)	
		880829	4249964	15.27 (15.27)	4.38 (26.34)		15.24 (15.31)	6.31 (28.30)	
		880829	4029944	11.09 (11.09)	3.61 (26.90)		10.99 (10.99)	4.66 (27.14)	
circuit5M_dc	4	880829	3302046	6.09 (6.09)	3.20 (26.90)	3	6.23 (6.22)	3.99 (27.70)	2
		880830	3283455	6.27 (6.27)	6.08 (26.29)		6.09 (6.09)	3.67 (28.30)	
		704663	3136583	12.24 (12.25)	4.04 (26.60)		12.24 (12.24)	10.08 (27.01)	
		704663	2907582	9.88 (9.88)	3.48 (26.59)		9.87 (9.87)	4.40 (26.95)	
	5	704663	2607145	6.22 (6.21)	3.15 (26.22)	2	6.14 (6.14)	3.83 (26.89)	2
		704663	3577997	4.97 (4.96)	3.05 (26.69)		4.94 (4.94)	3.51 (26.54)	
		704665	2636102	4.92 (9.92)	9.67 (26.69)		4.94 (4.94)	3.31 (27.02)	
		587219	3002895	10.11 (10.12)	3.81 (27.08)		10.07 (10.08)	9.23 (30.16)	
		587219	2748283	8.40 (8.39)	3.26 (27.04)		8.39 (8.39)	4.20 (30.26)	
		587219	2528726	6.81 (6.80)	3.18 (27.44)		6.77 (6.77)	3.79 (30.19)	
	6	587219	2207789	4.23 (4.24)	2.95 (27.83)	2	4.26 (4.25)	3.35 (30.19)	2
		587219	2182690	4.14 (4.15)	2.89 (28.23)		4.26 (4.25)	3.50 (28.99)	
		587222	2195026	4.26 (4.26)	5.57 (28.23)		4.14 (4.14)	3.27 (29.39)	

Three important aspects related to the parallel results should be highlighted.

(1) The proposed method uses a hybrid Jacobi Gauss-Seidel method as a smoother and a coarse system solver. This method applies the Gauss-Seidel method inside each processor and the Jacobi method between processors. So, as the number of processors increases, both smoother and coarse solvers become different. Also in the sequential version of the method this approach has not been reproduced. This can help us to explain the difference in the number of iterations in the parallel and sequential versions.

(2) The way in which the matrices are split between the processors (the same number of rows) may cause load unbalance in some cases that can affect the overall performance of the method. For the matrix *circuit5M_dc*, for example, the number of nonzero elements in the processor one is 25% larger than in the processor two, when 2 processors are used. A new way to divide the matrix among the processors should be investigated.

(3) Despite the use of a homogeneous Beowulf cluster, the type of network connection based on a switched fast Ethernet network shows to be an important bottleneck. So, the effects of the fast Ethernet connection on the results should be evaluated separately. In order to do so, the $time(p)$ required by the solver phase execution on p processors, in which there is interprocessors communication, was evaluated

considering a relative time in relation to the MPI π calculation example [25]. It means the $time(p)$ is defined as (6.1)

$$time(p) = \frac{wtime(p)}{\pi\text{-}time(p)}, \tag{6.1}$$

in which $wtime(p)$ is the total time spent in the setup phase, which includes the MPI communication time and is measured using the MPI function MPI_Wtime(), and $\pi\text{-}time(p)$ is the time spent by the MPI π example, both with p processors. As an example, Table 7 presents the values of $\pi\text{-}time(p)$, $wtime(p)$, $time(p)$ and $speedup(p)$, $p = 1, \ldots, 6$, for the matrix *2cubes_sphere*, which were used to create the illustration in Figure 6(a). The values of $\pi\text{-}time(p)$ were obtained as the mean of 5 runs. All the other results in Figure 6 were derived in a similar way.

As usual, the absolute speedup S_p is used for analyzing the parallel performance, and it is defined as (6.2)

$$S_p = speedup(p) = \frac{time(1)}{time(p)}, \tag{6.2}$$

in which $time(1)$ is the time spent by the best sequential algorithm and $time(p)$ as defined in (6.1).

As the MPI π example uses only point-to-point communication functions, the relative time approach can help to clarify if the apparent poor scalability is due to a high collective communication cost or mainly due to the type of network connection used.

The solver and preconditioner speedups are illustrated in Figure 6 for the matrices *circuit5M_dc*, *2cubes_sphere*, and *offshore*.

7. Conclusions

The PWAMG method, proposed in this work, has been applied as a black-box solver and preconditioner in some circuit simulations and finite element electromagnetic problems with good results.

An important characteristic of the proposed approach is its small demand for interprocessors communication. Actually, no communication is required in the setup phase if first-order filters are used. This characteristic is confirmed by the results for setup time presented in Tables 3–6, observing that the times measured by MPI_Wtime() and C clock() functions are practically the same. It is a great advantage of the proposed approach since this phase is the most time-consuming one.

In the solver phase, in which there is interprocessors communication, the numerical results seem to show a poor performance with more than 2 processors, even when the number of iterations does not increase by using more processors. These results can be caused in part due to the use of a collective communication function to update the vector after the matrix operations, which is motivated by the manner the matrix is divided between the processors.

In order to update the vector each processor should send and receive the part of the vector to all of the other processors. Of course, when the number of processors increases, this work also becomes larger.

Table 6: Parallel results for *dielFilterV2real* matrix (not enough memory).

	np	Nrows	Nonzeros	PWAMG solver		
				Setup time CPU (Wtime)	Solver time CPU (Wtime)	n
dielFilterV2real	1	1157456	48538952	—	—	—
	2	578728	26710406	60.00 (182.0)	176.21 (1763.4)	4
		578728	21828546	34.46 (39.34)	86.20 (1759.74)	
	3	385818	19476388	36.92 (78.71)	98.12 (192.62)	4
		385818	14529204	22.49 (22.54)	43.69 (192.13)	
		385820	14533360	22.18 (22.19)	54.49 (192.14)	
	4	289364	15807646	28.18 (28.39)	75.34 (108.75)	4
		289364	10902962	16.61 (16.59)	32.17 (108.35)	
		289364	10902760	16.79 (16.79)	33.26 (108.55)	
		289364	10925584	16.56 (16.58)	34.80 (108.75)	
	5	231491	13644000	24.42 (24.47)	64.90 (101.42)	4
		231491	8751322	13.28 (13.28)	27.28 (101.41)	
		231491	8727696	13.30 (13.29)	28.07 (101.57)	
		231491	8710580	13.35 (13.34)	26.93 (101.72)	
		231492	8705354	13.25 (13.25)	35.10 (101.72)	
	6	192909	11858182	22.81 (38.14)	107.36 (176.56)	7
		192909	7618206	11.76 (11.76)	47.08 (176.63)	
		192909	7295218	10.99 (10.99)	42.92 (176.76)	
		192909	7255410	11.18 (11.18)	42.60 (176.76)	
		192909	7233986	11.07 (11.07)	42.53 (176.50)	
		192911	7277950	11.01 (11.01)	58.37 (176.50)	

Table 7: Solver phase speedup using a relative *time*(p): example for matrix *2cubes_sphere.*

	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$
(A) *wtime*(p)	2.02	2.99	3.47	3.26	3.32	3.27
(B) π-*time*(p)	0.0000885	0.0002926	0.0003222	0.000997	0.001011	0.0009938
(C) *time*(p) = A/B	22824.86	10218.73	10769.71	3269.81	3283.88	3290.40
(D) *speedup*(p)	1.00	2.23	2.12	6.98	6.95	6.94

An alternative to overcome this problem may be to apply some graph partitioning method and develop an approach in which each processor should communicate only with its neighbors. Such approach is under development, and it is out of the scope of this paper.

However, the type of network connection based on a switched fast Ethernet network shows to be an important bottleneck, and its effects should be evaluated separately. As the MPI π example uses only point-to-point communication functions, the relative time approach can help to clarify if the apparent poor scalability is due to a high collective communication cost or mainly due to the type of network connection used. The speedup results based on the relative times show that the proposed approach is promising and that the kind of network connection that has been used is maybe the most important drawback.

Moreover, in spite of the speedup being less than the linear in some cases, it is important to mention an important aspect of this application: in the context of large sparse linear system of equations, where this paper is inserted, the problems have large memory

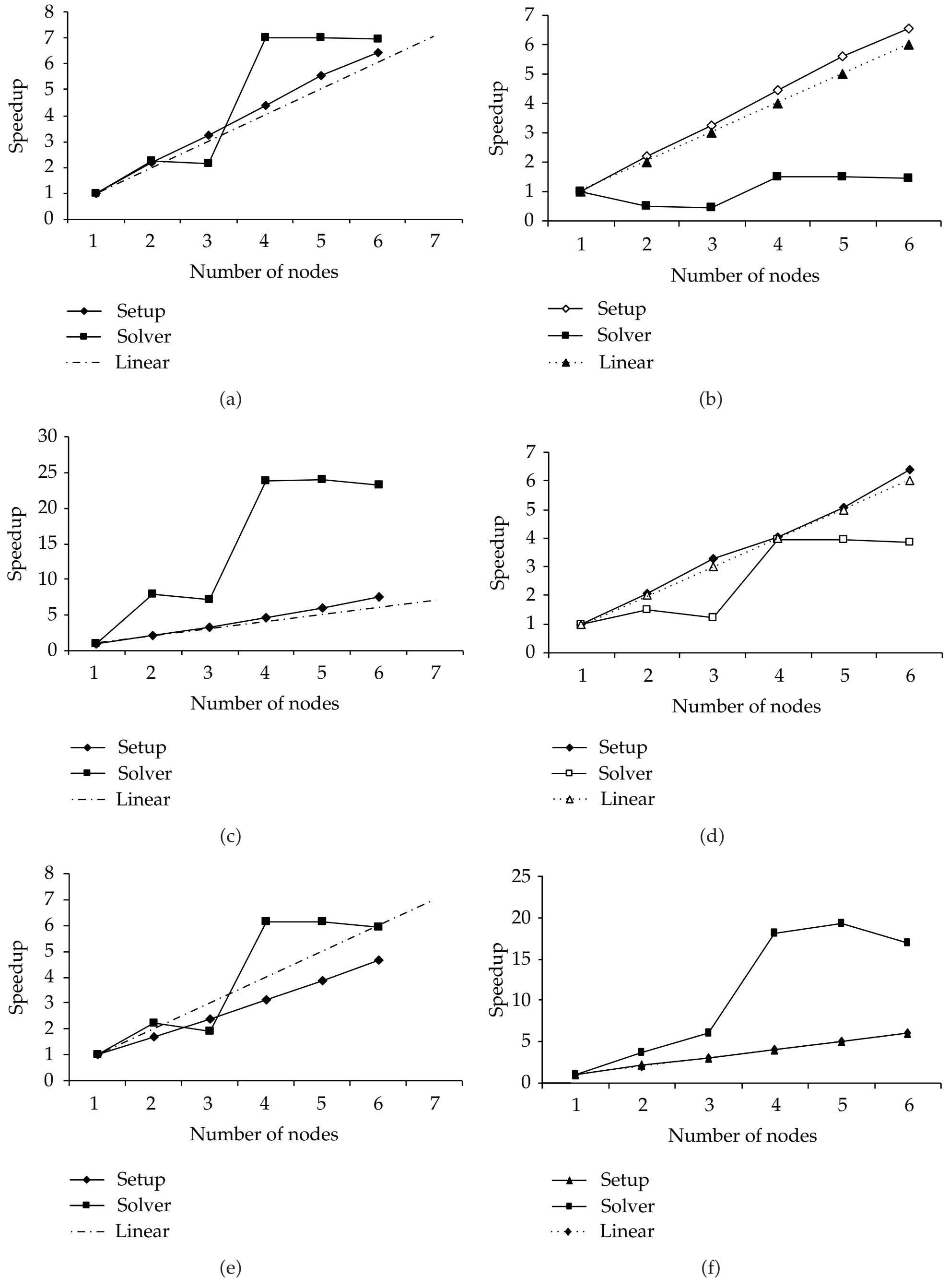


Figure 6: Speedup for *2cubes_sphere* solver (a) and preconditioner (b), *offshore* solver (c) and preconditioner (d), and *circuit5M_dc* solver (e) and preconditioner (f).

requirements. In these cases, as presented in [28], the speedup necessary to be cost effective can be much less than linear. The parallel program does not need p times memory of the unit processor, since parallelizing a job rarely multiplies its memory requirements by p.

Finally, the number of iteration of the proposed method seems to be independent of the number of processors. However, it is necessary to carry out more tests with a larger number of processors in order to draw more definitive conclusions. Nowadays, the authors are looking for new resources and/or partnerships to enable the continuation of this work.

Acknowledgments

Authors would like to thank the São Paulo Research Foundation (FAPESP) (06/59547-5 and 2011/06725-1) and Nove de Julho University (UNINOVE) for the financial support.

References

[1] G. Haase, M. Kuhn, and S. Reitzinger, "Parallel algebraic multigrid methods on distributed memory computers," *SIAM Journal on Scientific Computing*, vol. 24, no. 2, pp. 410–427, 2002.

[2] V. E. Henson and U. M. Yang, "BoomerAMG: a parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[3] H. de Sterck, U. M. Yang, and J. J. Heys, "Reducing complexity in parallel algebraic multigrid preconditioners," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 4, pp. 1019–1039, 2006.

[4] M. Griebel, B. Metsch, D. Oeltz, and M. A. Schweitzer, "Coarse grid classification: a parallel coarsening scheme for algebraic multigrid methods," *Numerical Linear Algebra with Applications*, vol. 13, no. 2-3, pp. 193–214, 2006.

[5] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, *Coarse-Grid Selection for Parallel Algebraic Multigrid*, Lawrence Livermore National Laboratory, Livermore, Calif, USA, 2000.

[6] L. Yu. Kolotilina and A. Yu. Yeremin, "Factorized sparse approximate inverse preconditionings. I. Theory," *SIAM Journal on Matrix Analysis and Applications*, vol. 14, no. 1, pp. 45–58, 1993.

[7] M. J. Grote and T. Huckle, "Parallel preconditioning with sparse approximate inverses," *SIAM Journal on Scientific Computing*, vol. 18, no. 3, pp. 838–853, 1997.

[8] D. Hysom and A. Pothen, "A scalable parallel algorithm for incomplete factor preconditioning," *SIAM Journal on Scientific Computing*, vol. 22, no. 6, pp. 2194–2215, 2000.

[9] P. Raghavan, K. Teranishi, and E. G. Ng, "A latency tolerant hybrid sparse solver using incomplete Cholesky factorization," *Numerical Linear Algebra with Applications*, vol. 10, no. 5-6, pp. 541–560, 2003.

[10] P. Raghavan and K. Teranishi, "Parallel hybrid preconditioning: incomplete factorization with selective sparse approximate inversion," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1323–1345, 2010.

[11] C. Janna, M. Ferronato, and G. Gambolati, "A block Fsai-Ilu parallel preconditioner for symmetric positive definite linear systems," *SIAM Journal on Scientific Computing*, vol. 32, no. 5, pp. 2468–2484, 2010.

[12] C. Janna and M. Ferronato, "Adaptive pattern research for block FSAI preconditioning," *SIAM Journal on Scientific Computing*, vol. 33, no. 6, pp. 3357–3380, 2011.

[13] M. Benzi and M. Tuma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, no. 2-3, pp. 305–340, 1999.

[14] F. H. Pereira, S. L. L. Verardi, and S. I. Nabeta, "A wavelet-based algebraic multigrid preconditioner for sparse linear systems," *Applied Mathematics and Computation*, vol. 182, no. 2, pp. 1098–1107, 2006.

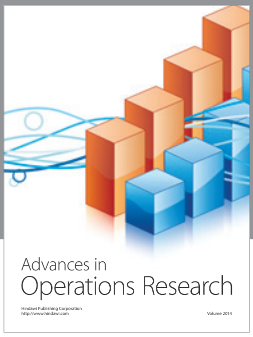
[15] F. H. Pereira, M. F. Palin, S. L. L. Verardi, V. C. Silva, J. R. Cardoso, and S. I. Nabeta, "A wavelet-based algebraic multigrid preconditioning for iterative solvers in finite-element analysis," *IEEE Transactions on Magnetics*, vol. 43, no. 4, pp. 1553–1556, 2007.

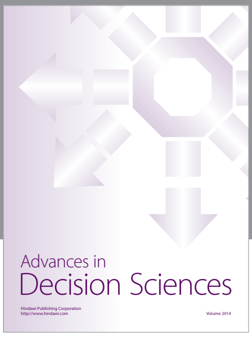
[16] F. H. Pereira, M. M. Afonso, J. A. De Vasconcelos, and S. I. Nabeta, "An efficient two-level preconditioner based on lifting for FEM-BEM equations," *Journal of Microwaves and Optoelectronics*, vol. 9, no. 2, pp. 78–88, 2010.

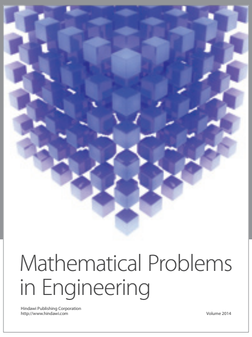
[17] T. K. Sarkar, M. Salazar-Palma, and C. W. Michael, *Wavelet Applications in Engineering Electromagnetics*, Artech House, Boston, Mass, USA, 2002.

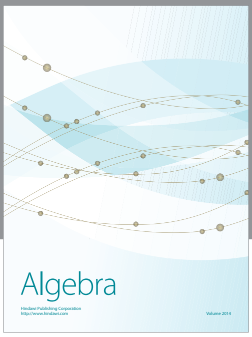
[18] V. M. Garcıa, L. Acevedo, and A. M. Vidal, "Variants of algebraic wavelet-based multigrid methods: application to shifted linear systems," *Applied Mathematics and Computation*, vol. 202, no. 1, pp. 287–299, 2008.

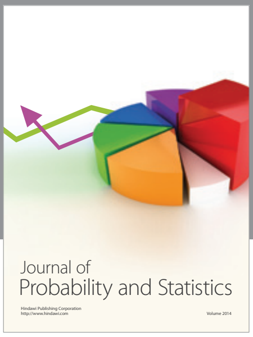
[19] A. Avudainayagam and C. Vani, "Wavelet based multigrid methods for linear and nonlinear elliptic partial differential equations," *Applied Mathematics and Computation*, vol. 148, no. 2, pp. 307–320, 2004.
[20] D. de Leon, "Wavelet techniques applied to multigrid methods," CAM Report 00-42, Department of Mathematics UCLA, Los Angeles, Calif, USA, 2000.
[21] G. Wang, R. W. Dutton, and J. Hou, "A fast wavelet multigrid algorithm for solution of electromagnetic integral equations," *Microwave and Optical Technology Letters*, vol. 24, no. 2, pp. 86–91, 2000.
[22] R. S. Chen, D. G. Fang, K. F. Tsang, and E. K. N. Yung, "Analysis of millimeter wave scattering by an electrically large metallic grating using wavelet-based algebratic multigrid preconditioned CG method," *International Journal of Infrared and Millimeter Waves*, vol. 21, no. 9, pp. 1541–1560, 2000.
[23] F. H. Pereira and S. I. Nabeta, "Wavelet-based algebraic multigrid method using the lifting technique," *Journal of Microwaves and Optoelectronics*, vol. 9, no. 1, pp. 1–9, 2010.
[24] J. M. Ford, K. Chen, and N. J. Ford, "Parallel implementation of fast wavelet transforms," Numerical Analysis 39, Manchester University, Manchester, UK, 2001.
[25] H. J. Sips and H. X. Lin, *High Performance Computing Course MPI Tutorial*, Delft University of Technology Information Technology and Systems, Delft, The Netherlands, 2002.
[26] F. Alessandri, M. Chiodetti, A. Giugliarelli et al., "The electric-field integral-equation method for the analysis and design of a class of rectangular cavity filters loaded by dielectric and metallic cylindrical pucks," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 8, pp. 1790–1797, 2004.
[27] T. A. Davis, "Algorithm 849: a concise sparse Cholesky factorization package," *ACM Transactions on Mathematical Software*, vol. 31, no. 4, pp. 587–591, 2005.
[28] D. A. Wood and M. D. Hill, "Cost-effective parallel computing," *Computer*, vol. 28, no. 2, pp. 69–72, 1995.

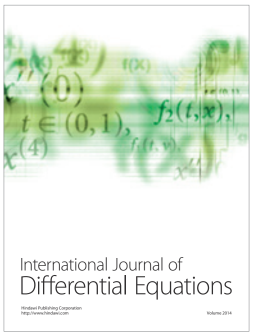












Hindawi

Submit your manuscripts at
http://www.hindawi.com