

Adaptive Calculation of Variable Coefficients Elliptic Differential Equations via Wavelets

Amir Averbuch¹, Leonid Beliak², Moshe Israeli^{2*}

¹School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978
Israel

² Faculty of Computer Science, Technion- Israel Institute of Technology
Haifa 32000, Israel

Abstract

We propose a solver for 1-D strictly elliptic linear PDE's with non-constant coefficients of the form $\Delta U - b(x)U = f(x)$. We combine a sparse multiplication algorithm with a diagonally preconditioned conjugate gradient (CG) method. We use sparse data structures to take advantage of the $O(N_s)$ complexity of the algorithm, where N_s is the number of significant coefficients (i.e. above a certain threshold) required for a given accuracy.

We show that the usage of a sparse multiplication in wavelet space rather than in the original physical space can speed up the performance of the sparse solver by a factor of 20. We present an algorithm and numerical results for an adaptive multiplication scheme that can rapidly solve the equation above. We explore, in detail, how the accuracy of the wavelet-based multiplication is affected by different input parameters for the algorithm. We integrated a sparse multiplication into the PDE solver. The relation between the performance of the solver and the parameters of the wavelet based sparse multiplication is also studied. This integration allowed us to extend the fast adaptive algorithms to achieve numerical solutions of linear non-constant coefficient differential equations. One-dimensional numerical examples for using stand alone sparse multiplication and for the differential equations solver are presented. This work, which is based on [1], extends it.

*Prof. Moshe Israeli passed away on February 18, 2007. This paper is dedicated to his memory.

1 Introduction

1.1 General Description

Generating a “good” discrete representation for continuous operators is one of the basic problems in numerical solution of differential and integral equations. Classical methods for discretization lead to a dense representation for most operators. An important step towards efficient computation is to construct sparse representations of common operators and fast algorithms for using them.

Numerical algorithms using wavelet bases are similar to other transform methods where vectors and operators are projected on a basis and then the computations take place in the new system of coordinates. This approach has an advantage if in the new system of coordinates the computation is faster than in the original coordinates.

Wavelet-based calculations have a good localization of significant wavelet coefficients both in space and wavelet domains. Thus, a sparse wavelet representation may be generated for certain classes of functions if, additionally, one allows for automatic adaptivity by using thresholding. The result of this approach is that only a few coefficients are needed to describe a smooth section of the right hand side (the forcing function) while more coefficients are needed for sharp transitions and points of singularity. This is a result of the vanishing moments property of high order wavelets.

Furthermore, wide spread classes of operators in wavelet bases (for example differential and pseudo differential operators) are reduced to a sparse form. They can be translated to fast schemes for solving numerical problems [1, 2, 3, 8, 10] as will be explained below.

We consider efficient discretization and solution of differential equations with non-oscillatory behavior that has possibly localized regions with irregular structures. Similar to other discretization schemes for the solution of PDEs, in wavelet schemes we have to solve a system of linear equations in the wavelet coordinate basis. Conjugate gradient method is one of the most efficient methods to solve such a system, but the convergence rate of this method may be too slow unless the condition number of the matrix (which represents the system) is small enough. Therefore, efficient implementation of the conjugate gradient method requires a use of some preconditioner. In general, application of a preconditioner to the above-mentioned matrix can be expensive. In [1, 2, 3], a diagonal preconditioner was presented (for matrices generated by wavelet-transform of differential operators) which renders the condition number to $O(1)$ and which can be applied to the matrix with low cost. Therefore, by using this preconditioned conjugate gradient method, only a constant number of

steps is needed to obtain a solution with a prescribed accuracy.

The works [1, 2, 3, 10] present an efficient approach for solving PDEs with constant coefficients. Other works, such as [12], analyzed wavelet methods (for example convergence rates, approximation errors etc.) from a more theoretical point of view.

Let

$$\frac{d^2U}{dx^2} - b(x)U = f(x). \quad (1.1)$$

After the application of wavelet based functions with sufficient number of vanishing moments on the right hand side of Eq. 1.1, we may get a sparse representation of smooth areas. Let N_s be the number of significant coefficients remaining after thresholding. N_s can be much lower than N where N is the number of data points on the grid. In these cases, only N_s coefficients should be updated, resulting in a $O(N_s)$ algorithm. The proposed algorithms will have better performance as long as we have $N_s \ll N$. In equations such as Eq. 1.1 it is insufficient to achieve better performance by having a sparse representation for the solution U , the right-hand side of the equation and the differential operator. Since the multiplication $b(x)U$, which is being done in “real” (physical) space, can significantly increase the run time if it is performed as dense multiplication. Scheme for sparse multiplication in wavelet space was described in [1, 10]. We extended this scheme, integrated differential equation solver and show how the usage of masks on the high pass channel of the wavelet multiscale decomposition (called s-masks) can help in speeding up the solution.

In this work, we show that the usage of sparse multiplication in the wavelet space rather than in its original physical space can speed up the computation by a factor of 20 over a regular solver. Therefore, we present an algorithm and numerical results for adaptive multiplication scheme that can solve Eq. (1.1) fast. We explore how the dependency of the accuracy of the wavelet-based multiplication is affected by different input parameters of the algorithm. We extend and integrate sparse multiplication into a PDE solver. The relation between the performance of the solver and the parameters to the wavelet based sparse multiplication is studied. This integration allows us to extend the above fast adaptive algorithms to numerical solution of linear non-constant coefficients differential equations.

The paper has the following structure: In section 2.4, we present the theory of evaluation of functions in wavelet basis. We describe there the adaptive multiplication algorithm and analyze its accuracy. In section 3, we describe an adaptive PDE solver and analyze how the adaptive multiplication affects the accuracy and the run-time of the proposed solver.

2 Adaptive Multiplication

2.1 Introduction to Wavelet Analysis

In this section, we review the relevant material associated with wavelet basis expansions of functions and operators. Much of this material appeared in a number of publications, see e.g. [15, 17, 18] for more details. We consider a multiresolution analysis (MRA) of $\mathbf{L}^2(\mathbb{R})$ as

$$\dots \subset \mathbf{V}_2 \subset \mathbf{V}_1 \subset \mathbf{V}_0 \subset \mathbf{V}_{-1} \subset \mathbf{V}_{-2} \subset \dots, \quad (2.1)$$

We define an associated sequence of subspaces \mathbf{W}_j as the orthogonal complements of \mathbf{W}_j in \mathbf{V}_{j-i}

$$\mathbf{V}_{j-1} = \mathbf{V}_j \oplus \mathbf{W}_j. \quad (2.2)$$

Repeatedly using (2.2) shows that subspace \mathbf{V}_j can be rewritten as the direct sum

$$\mathbf{V}_j = \bigoplus_{j' > j} \mathbf{W}_{j'}. \quad (2.3)$$

We denote by $\varphi(\cdot)$ the scaling function and by $\psi(\cdot)$ the wavelet function. The family of functions $\{\varphi_{j,k}(x) = 2^{-j/2}\varphi(2^{-j}x - k)\}_{k \in \mathbb{Z}}$ forms an orthonormal basis of \mathbf{V}_j and the family of functions $\{\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k)\}_{k \in \mathbb{Z}}$, forms an orthonormal basis of \mathbf{W}_j . The function φ may be expressed as a linear combination of the basis functions of \mathbf{V}_{-1} ,

$$\varphi(x) = \sqrt{2} \sum_{k=1}^{L_f} h_k \varphi(2x - k). \quad (2.4)$$

Similarly, we have

$$\psi(x) = \sqrt{2} \sum_{k=0}^{L_f-1} g_k \psi(2x - k). \quad (2.5)$$

The coefficients $H = \{h_k\}_{k=0}^{L_f-1}$ and $G = \{g_k\}_{k=0}^{L_f-1}$ are the quadrature mirror filters (QMF's) of length L_f .

For numerical purposes, we define the 'finest' scale $j = 0$, and the 'coarsest' scale $j = J$, such that the infinite chain (2.1) is restricted to

$$\mathbf{V}_J \subset \mathbf{V}_{J-1} \subset \dots \subset \mathbf{V}_0, \quad (2.6)$$

where the subspace \mathbf{V}_0 is finite dimensional. Specifying the QMFs H and G in numerical experiments defines the properties of the wavelet basis. We will also consider

a periodized version of the multiresolution analysis that is obtained if we consider periodic functions. Such functions then have projections on \mathbf{V}_0 which are periodic of period $N = 2^n$, where N is the dimension of \mathbf{V}_0 . With a slight abuse of notation we will denote these periodized subspaces also by \mathbf{V}_j and \mathbf{W}_j . We can then view the space \mathbf{V}_0 as consisting of 2^n ‘samples’ or lattice points and each space \mathbf{V}_j and \mathbf{W}_j as consisting of 2^{n-j} lattice points, for $j = 1, 2, \dots, J \leq n$.

2.2 Representation of functions in Wavelet Basis

The projection of a function $f(x)$ onto subspace \mathbf{V}_j is given by

$$(P_j f)(x) = \sum_{k \in \mathbb{Z}} s_k^j \varphi_{j,k}(x), \quad (2.7)$$

where P_j denotes the projection operator onto the subspace \mathbf{V}_j . The set of coefficients $\{s_k^j\}_{k \in \mathbb{Z}}$, which we refer to as ‘averages’, is computed via the inner product

$$s_k^j = \int_{-\infty}^{+\infty} f(x) \varphi_{j,k}(x) dx. \quad (2.8)$$

Alternatively, it follows from (2.3) and (2.1) that we can also write $(P_j f)(x)$ as a sum of projections of $f(x)$ onto subspaces $\mathbf{W}_{j'}, j' > j$

$$(P_j f)(x) = \sum_{j' > j} \sum_{k \in \mathbb{Z}} d_k^{j'} \psi_{j',k}(x), \quad (2.9)$$

where the set of coefficients $\{d_k^j\}_{k \in \mathbb{Z}}$, which we refer to as ‘differences’, is computed via the inner product

$$d_k^j = \int_{-\infty}^{+\infty} f(x) \psi_{j,k}(x) dx. \quad (2.10)$$

The projection of a function onto a subspace \mathbf{W}_j is denoted $Q_j f(x)$, where $Q_j = P_{j-1} - P_j$. Since we compute a ‘periodized’ MRA, on each subspace \mathbf{V}_j and \mathbf{W}_j the coefficients of the projection satisfy

$$s_j^k = s_{k+2^{n-j}}, \quad d_j^k = d_{k+2^{n-j}} \quad (2.11)$$

for each $j = 1, 2, \dots, J$ and $k \in \mathbb{F}_{2^{n-j}} = \mathbb{Z}/2^{n-j}\mathbb{Z}$, where $\mathbb{F}_{2^{n-j}}$ is the finite field of 2^{n-j} integers, e.g. the set $\{0, 1, \dots, 2^{n-j} - 1\}$.

In our numerical algorithms, the expansion into wavelet basis of $(P_0 f)(x)$ is given by sum of successive projection on subspaces \mathbf{W}_j , $j = 1, 2, \dots, J$, and a final ‘coarse’ scale projection on \mathbf{V}_j ,

$$(P_0 f)(x) = \sum_{j=1}^J \sum_{k \in \mathbb{F}_{2^{n-j}}} d_k^j \psi_{j,k}(x) + \sum_{k \in \mathbb{F}_{2^{n-j}}} s_k^J \varphi_{J,k}(x). \quad (2.12)$$

2.3 Evaluation of functions in wavelet bases

This section describes a method for pointwise product evaluation of functions represented in wavelet bases. More generally, these results can be used to compute functions $f(u)$, where f is an analytic function and u is expanded in a wavelet basis. Since pointwise multiplication is a diagonal operator in the ‘physical’ space, computation of the pointwise product in any other domain appears to be less efficient. A successful and efficient algorithm should at some point compute $f(u)$ in the physical space using values of u and not its expansion. Here are several observations regarding the calculation of $f(u)$, where u is expanded in an arbitrary basis $u(x) = \sum_{i=0}^N u_i b_i(x)$ where u_i are the coefficients and $b_i(x)$ are the basis functions. In general, we have $f(u(x)) \neq \sum_{i=0}^N f(u_i) b_i(x)$.

For example, if $u(x)$ is expanded in its Fourier series, clearly the Fourier coefficients of the function $f(u)$ do not correspond to the function of the Fourier coefficients. This led to the development of pseudo-spectral algorithms for numerical solutions of partial differential equations. In order to explain the algorithm that computes $f(u)$ in the wavelet system of coordinates, we begin with the assumption that u and $f(u)$ are both elements of \mathbf{V}_0 , i.e. $u, f(u) \in \mathbf{V}_0$. Then,

$$u(x) = \sum_k s_k^0 \varphi(x - k) \quad (2.13)$$

where s_k^0 are the coefficients which are defined by

$$s_k^0 = \int_{-\infty}^{\infty} u(x) \varphi(x - k) dx. \quad (2.14)$$

Let us assume that the scaling function is interpolating, so that

$$s_k^0 = u(k). \quad (2.15)$$

Since $u, f(u) \in \mathbf{V}_0$ we obtain

$$f(u) = \sum_k f(s_k^0) \varphi(x - k). \quad (2.16)$$

In section 2.4 we describe an adaptive algorithm for computing the pointwise multiplication of the function $f(u) = u^2$. In this algorithm, we split $f(u)$ into projections on different subspaces. Then, we consider ‘pieces’ of the wavelet expansion of u in finer subspaces where we calculate the contributions to $f(u)$ using an approximation to (2.16). This is a direct comparison with the calculation of $f(u)$ in a basis where the entire expansion must first be projected into a ‘physical’ space where $f(u)$ is then computed, e.g. using pseudo-spectral methods.

2.4 Adaptive algorithm for calculation of u^2

Since the product of two functions can be expressed as a difference of squares, it is sufficient to present an algorithm that evaluates u^2 . The algorithm we used is a modification of the algorithm which was described in [10]. In order to compute u^2 in a wavelet basis, we first recall that the projections of u on the subspaces \mathbf{V}_j and \mathbf{W}_j are given by $P_j(u) \in \mathbf{V}_j$ and $Q_j(u) \in \mathbf{W}_j$ for $j = 0, 1, 2, \dots, J \leq n$, respectively, where J is the last level (scale) from where the coefficients will be taken to be used in the wavelet multiplication. Let $j_f, 1 \leq j_f \leq J$, be the finest scale having significant wavelet coefficients that are greater than ε . This produces an ε -accurate approximation of u . Therefore, from Eq. (2.12), the projection of u can be expressed

$$\text{as } (P_0u)(x) \approx \sum_{j=j_f}^J \sum_{\{k:|d_k^j|>\varepsilon\}} d_k^j \psi_{j,k}(x) + \sum_{0 \leq k < 2^{n-J}} s_k^J \varphi_{J,k}(x).$$

Let us consider the case where u and $u^2 \in \mathbf{V}_0$, so that we can expand $(P_0u)^2$ in a ‘‘telescopic’’ series:

$$(P_0u)^2 - (P_Ju)^2 \approx \sum_{j=j_f}^J (P_{j-1}u)^2 - (P_ju)^2. \quad (2.17)$$

Decoupling scale interactions in (2.17) using $P_{j-1} = Q_j + P_j$, we arrive at

$$(P_0u)^2 \approx (P_Ju)^2 + \sum_{j=j_f}^J 2(P_ju)(Q_ju) + (Q_ju)^2. \quad (2.18)$$

Since the scaling function is interpolating and $u, u^2 \in \mathbf{V}_0$ then $(P_0u)^2$ is actually (P_0u^2) .

Remark: Equation (2.18) is written with terms from a finite number of scales. If $j \in \mathbb{Z}$, then (2.18) can be rewritten as

$$u^2 \approx \sum_{j \in \mathbb{Z}} 2(P_ju)(Q_ju) + (Q_ju)^2. \quad (2.19)$$

Evaluation of (2.18) requires to compute $(Q_j u)^2$ and $(Q_j u)(P_j u)$, where $Q_j u$ and $P_j u$ are elements of the subspaces on the same scale and, thus, have basis functions with the same support size. In addition, we need to compute $(Q_j u)^2$ which involves only the coarsest scale and therefore it is not computationally expensive. The difficulty in evaluating (2.18) lies in the fact that the terms $(Q_j u)^2$ and $(Q_j u)(P_j u)$ do not necessarily belong to the same subspace as the multiplicands. However, since

$$\mathbf{V}_j \oplus \mathbf{W}_j = \mathbf{V}_{j-1} \subset \mathbf{V}_{j-2} \subset \dots \mathbf{V}_{j-j_0} \subset \dots \quad (2.20)$$

we may think of both $P_j u \in \mathbf{V}_j$ and $Q_j u \in \mathbf{W}_j$ as elements of a finer subspace, which is denoted by \mathbf{V}_{j-j_0} for some $j_0 \geq 1$

The coefficients $s_k^{-j_0}$ of the projection of u on \mathbf{V}_{-j_0} are given by:

$$s_k^{-j_0} = 2^{j_0/2} \int_{-\infty}^{\infty} u(x) \varphi(2^{j_0} x - k) dx. \quad (2.21)$$

We compute $Q_j u$ and $P_j u$ in \mathbf{V}_{j-j_0} using wavelet reconstruction. On \mathbf{V}_{j-j_0} , we can calculate the contribution to (2.18) using (2.21).

It is important that in order to apply (2.21) we may always choose j_0 in such a way that, to within a given accuracy ε , $(Q_j u)^2$ and $(Q_j u)(P_j u)$ belong to \mathbf{V}_{j-j_0} (see proof in [10]). In practice, j_0 must be small in order to get fast computations. In our numerical experiments j_0 was 2, 3 and it is also possible to use even $j_0 = 1$.

To describe the algorithm which computes the pointwise product, let us denote by $\mathcal{R}_j^{j_0}(\cdot)$ the operator to reconstruct (represent) a vector on subspace \mathbf{V}_j or \mathbf{W}_j in the subspace \mathbf{V}_{j-j_0} . In the subspace \mathbf{V}_{j-j_0} , we can then use the coefficients of $\mathcal{R}_j^{j_0}(P_j u)$ and $\mathcal{R}_j^{j_0}(Q_j u)$ to calculate the contribution to the product (2.18) using ordinary multiplication as in (2.21). To this end, the contributions to (2.18), for $j = j_f, j_f + 1, \dots, J - 1$, are computed as:

$$\mathcal{P}_{j-j_0}(u^2) = 2(\mathcal{R}_{j_0}^j(P_j u))(\mathcal{R}_{j_0}^j(Q_j u)) + (\mathcal{R}_{j_0}^j(Q_j u))^2, \quad (2.22)$$

where $\mathcal{P}_j(f(u))$ is the contribution to $f(u)$ on the subspace \mathbf{V}_j . On the final coarsest scale J , we compute:

$$\mathcal{P}_{J-j_0}(u^2) = (\mathcal{R}_{j_0}^J(P_J u))^2 + 2(\mathcal{R}_{j_0}^J(P_J u))(\mathcal{R}_{j_0}^J(Q_J u)) + (\mathcal{R}_{j_0}^J(Q_J u))^2. \quad (2.23)$$

Notation

u	- input vector
u^2	- output: calculation of the pointwise multiplication $u \cdot u$.
h and g	- low and high pass filters (see Eqs. 2.4 and 2.5, respectively)
wr_x	- wavelet representation of the vector x
$wr_x \cdot P_j$	- projection of the vector x on the subspace \mathbf{V}_j
$wr_x \cdot Q_j$	- projection of the vector x on the subspace \mathbf{W}_j
$P_{1,2,3}Temp_j$	- temporary variables that hold the projections on the subspace \mathbf{V}_j
\xrightarrow{r}	- reconstruction operation for one level
\xrightarrow{d}	- decomposition operation for one level

A flow description of the implementation of the algorithm

1. A given vector $u \rightarrow wr_u$ is decomposed (wavelet representation of u)
2. Begin at level j_f ($j = j_f, \dots, J$)
 - (a) $wr_u \cdot P_j \xrightarrow{r} P_1Temp_{j-1}$: reconstruction with the h filter ($V_j \xrightarrow{r} V_{j-1}$)
 - (b) $wr_u \cdot Q_j \xrightarrow{r} P_2Temp_{j-1}$: reconstruction with the g filter ($W_j \xrightarrow{r} V_{j-1}$)
 - (c) Repeat for $j_0 - 1$ times: $k = 1 \dots j_0 - 1$
 - i. $P_1Temp_{j-k} \xrightarrow{r} P_1Temp_{j-k-1}$ reconstruction with the h filter
($V_{j-k} \xrightarrow{r} V_{j-k-1}$)
 - ii. $P_2Temp_{j-k} \xrightarrow{r} P_2Temp_{j-k-1}$ reconstruction with the h filter
($V_{j-k} \xrightarrow{r} V_{j-k-1}$)
 - (d) The obtained $P_1Temp_{j-j_0}$ and $P_2Temp_{j-j_0}$ are $\mathcal{R}_j^{j_0}(P_j)$ and $\mathcal{R}_j^{j_0}(Q_j)$, respectively, and $P_3Temp_{j-j_0}$ is $\mathcal{P}_{j-j_0}(P_j)$ from Eqs. 2.22 (or 2.23). Therefore, we calculate i. (if $j = J$) or ii. (if $j \neq J$):
 - i. $P_3Temp = 2 \cdot (P_1Temp_{j-j_0} \cdot P_2Temp_{j-j_0}) + (P_2Temp_{j-j_0})^2$
from Eq. (2.22)
 - ii. $P_3Temp = 2 \cdot (P_1Temp_{j-j_0} \cdot P_2Temp_{j-j_0}) + (P_2Temp)^2$
 $+ (P_1Temp_{j-j_0})^2$ from Eq. (2.23)
 - (e) Repeat for $j_0 - 1$ times: $k = j_0 - 1$:
 - i. $P_3Temp_{j-k} \xrightarrow{d} P_3Temp_{j-k+1}$ decomposition with the h filter
($V_{j-k} \xrightarrow{d} V_{j-k+1}$)

(f) Calculate Q_j and P_j of wr_u^2 by :

- i. $P_3Temp_{j-1} \xrightarrow{d} wr_u^2.P_j$ (unscaled) decomposition with the h filter
($V_{j-1} \rightarrow V_j$)
- ii. $P_3Temp_{j-1} \xrightarrow{d} wr_u^2.Q_j$ (unscaled) decomposition with the g filter
($V_{j-1} \xrightarrow{d} W_j$)

3. Recalculate $wr_u^2.Q_j$ and $wr_u^2.P_j$ from the *unscaled* $wr_u^2.P_j$ and $wr_u^2.Q_j$ using a similar process as was done at the end of the “application of the non-standard form” from $wr_u^2.P_j$ and $wr_u^2.Q_j$ (see ?? and ?? in the appendix).

A specific example, which demonstrates all the steps that were involved in the application of the sparse multiplication algorithm with the parameters $j_f = 6$, $j_0 = 3$ and $J = 8$, is shown in Fig. 2.1. Gray areas indicate the existence of significant coefficients. At the left side of the figure we see the wavelet representation wr_u of the algorithm’s input vector u . The numbers 1 to 9 are the scales of the decomposition. The red arrows point to successive applications of steps 2.a and then $j_0 - 1$ applications of step 2.c.i. $\mathcal{R}_{(j)}^{j_0}(P_j u)$ is calculated from $wr_u.P$. The results are stored in P_1Temp_j . Similarly, the blue arrows point to successive applications of steps 2.b and then $j_0 - 1$ applications of step 2.c.ii. $\mathcal{R}_{(j)}^{j_0}(P_j u)$ is calculated from $wr_u.Q$. The results are stored in P_2Temp_j (while $j_f \leq j \leq J$). Afterwards, steps 2.d.i and 2.d.ii calculate $\mathcal{P}_{j-j_0}(u^2)$ by applying Eqs. 2.22 and 2.23. The results are stored in P_3Temp . The $wr_u^2.unscaled$ is calculated from P_3Temp through steps 2.e, 2.f.i and 2.f.ii. In the current context, “unscaled” means the following: steps 2.e and 2.f calculate the coefficients of the projections u^2 on all subspaces \mathbf{V}_j and \mathbf{W}_j ($j_f \leq j \leq J$), but the results should be stored as coefficients of the projections on \mathbf{V}_j ($j_f \leq j \leq J$) and \mathbf{W}_J . Therefore, we should recompute the coefficients of the projection on “unnecessary” \mathbf{W}_j ($j_f \leq j \leq J - 1$) through projections on \mathbf{V}_j and \mathbf{W}_J ($j_f + 1 \leq j \leq J$). Step 3 performs this calculation to construct wr_{u^2} .

In the original algorithm [8], the sparse multiplication began from some initial level (denoted by j_f). In this approach, the computation began at a level lower than 1. This means, that some initial finer level in the decomposition was ignored. Therefore, the accuracy of the algorithm was degraded. But taking into account the finer levels means degradation in the performance of the algorithm. The finer levels have substantial influence on the performance and on the accuracy. In this implementation, we cannot reconstruct more than one level if we start from scale 1. In our proposed solution, we consider all scales while starting at $j_f = 1$.

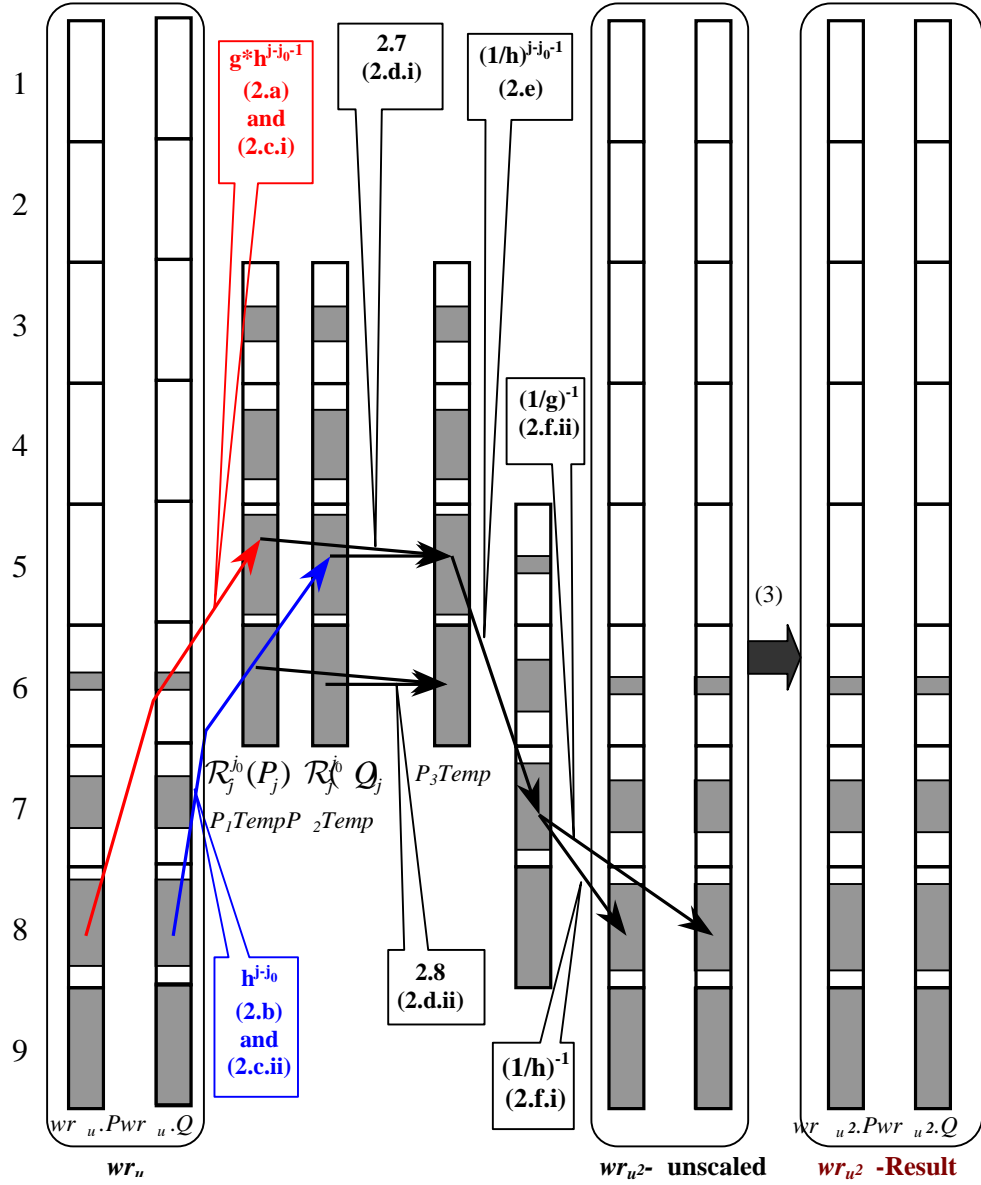


Figure 2.1: Algorithm for sparse computation of u^2 .

We propose a constructive solution for the above performance and accuracy problems while allowing all the levels to participate in the computation. In order to overcome the problem of not being able to reconstruct more than one level, we choose a different j_0 for different values of j (j is the current level in the algorithm). In steps 2.c.i and 2.c.ii of the algorithm, we reconstruct up to the finest level if $j < j_0$. For example, let assume that there are 7 levels in the wavelet representation of u , denoted as wr_u . The parameter j_0 is set to 2. Thus, we do not apply step 2.c.i and 2.c.ii in level $j = 1$ at all. In level $j = 2$ we apply steps 2.c.i and 2.c.ii once. For all other levels $j = 3, \dots, 7$, we apply steps 2.c.i and 2.c.ii exactly j_0 times. Sparse multiplication enables to overcome the problem that processing the first level decreases the performance of a stand-alone multiplication. Masking of the s and d coefficients (as described in Section 3.1 and in [1].) speeds-up the computation even if the finer level is processed. Masking allows us to update only the coefficients that have substantial influence on the right hand side of the differential equation.

2.5 Numerical results

We analyze here the efficiency and accuracy of the proposed adaptive multiplication $u \cdot u$, where u is a vector of $2^{14} = 16384$ points, that is generated by:

$$\begin{aligned} i &= 0, \dots, N - 1 \\ x &= -.5 + i / (N - 1) \\ u[i] &= x \cdot e^{-256.0 \cdot x^2} + 20.0 \cdot (0.2 + x) \cdot e^{-32000.0 \cdot (0.2 + x)^2}. \end{aligned}$$

u and u^2 are given in Fig. 2.2.

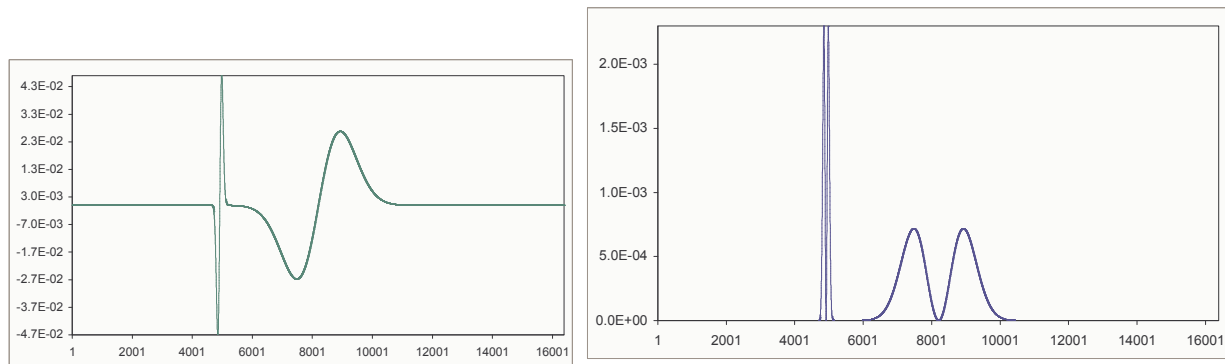


Figure 2.2: Left: The vector u that is being used as an input for the multiplication program. Right: Exact result for u^2

The dependency of the accuracy of the multiplication algorithm on different parameters is explained next. Actually, there are four numerical parameters that affect the algorithm's accuracy: N is the number of points and $\log_2 N$ is the number of levels in the wavelet decomposition of u , j_0 is the number of levels in the partial reconstruction before the application of the real multiplication, j_f is the starting level for the multiplication, and J is the final level of the multiplication $J \leq \log_2 N$. If $J < \log_2 N$ then before the application of the algorithm we perform a partial reconstruction of wr_u , which is the wavelet representation of u . Therefore, the final level of wr_u will be J . In Fig. 2.3, we can see the dependency of the sparse multiplication accuracy on j_0 . We can see that even for j_0 such as 2 and 3 we get satisfactory accuracy.

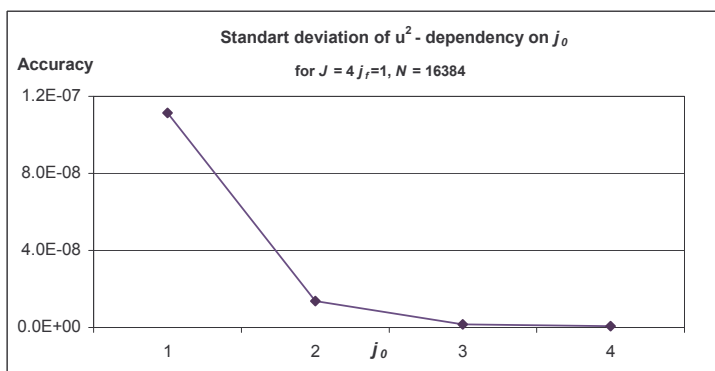


Figure 2.3: Dependency of the sparse multiplication the accuracy on j_0

This feature of the algorithm is extremely important. Recall that we are using a partial reconstruction because we have to compute $(Q_j u)^2 + (Q_j u)(P_j u)$ (see Eq. (2.18)). The problem is that the terms do not necessarily belong to the same subspace as the multiplicands, therefore, we use a partial reconstruction to transform the multiplicands to the same subspace and to use pointwise multiplication. Therefore, we can expect that after some specific j_0 , the achieved accuracy should fit our threshold (see [10]). But the fact that j_0 is relatively small allows us to apply this algorithm while we begin from the first scale. This provides us with an option to choose a more accurate but still a fast solution scheme for the differential equation.

Figure 2.4 shows how the accuracy is affected by j_f and J .

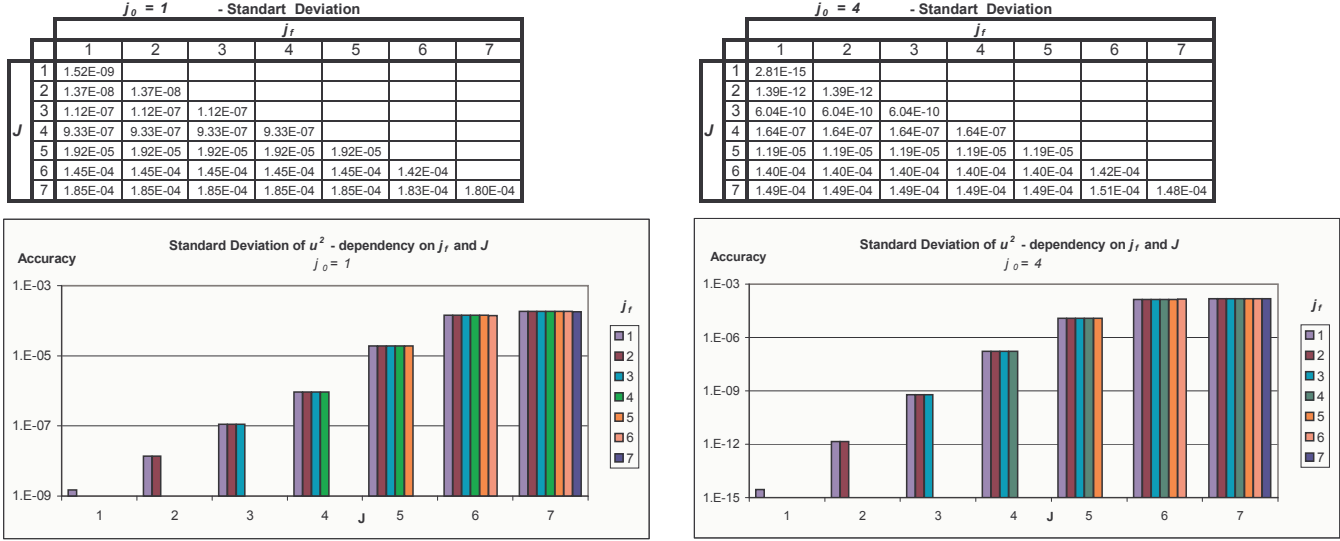


Figure 2.4: Dependency of the sparse multiplication accuracy on j_f and J . Left: $j_0 = 1$. Right: $j_0 = 4$

For low levels of J ($J \leq 7$) the accuracy is almost not affected by j_f . This happens because for small J we are still close to physical space and the “differences” in the lower levels are insignificant to approximate u . Therefore, we can just use the last level of the wavelet decomposition of u and still have a good accuracy.

This dependency of the accuracy on the parameters means that we can use relatively small j_f and J to be either 4 or 5 or 6. If we choose $j_0 = 2$ we obtain a satisfactory accuracy. We can also expect good speed-up in the differential equation solver. This is demonstrated in section 3.

3 Differential equations solver

3.1 Outline of the 1-D PDE solver

In this section, we describe a method for the solution of elliptic PDEs using wavelet basis. This is a fast adaptive method for solving certain elliptic equations with periodic boundary conditions. Let us consider the partial differential equation

$$\mathcal{L}u = f \quad x \in \mathbf{D} \subset \mathbf{R}^d, \tag{3.1}$$

with the boundary condition $\mathcal{B}u|_{\partial\mathbf{D}} = g$, where \mathcal{L} is an elliptic operator, $\mathcal{L}u = -\sum_{i,j=1,\dots,d} (a_{ij}(x) u_{x_i})_{x_j} + b(x) u$, and \mathcal{B} is the boundary operator, $\mathcal{B}u = \alpha u + \beta \frac{\partial u}{\partial N}$.

We generate a function f_{ext} , which is a smooth extension of f , outside the domain \mathbf{D} , such that f_{ext} is compactly supported in a rectangular box \mathbf{B} , $\mathbf{D} \subset \mathbf{B} \subset \mathbf{R}^d$, and $f = f_{ext}$ for $x \in \mathbf{D}$. We want to devise an adaptive efficient method to solve

$$\mathcal{L}u = f_{ext} \quad x \in \mathbf{B} \quad (3.2)$$

with periodic boundary conditions. Let us illustrate for simplicity the solution strategy on Poisson's equation

$$\Delta u = f \quad x \in \mathbf{B} \quad (3.3)$$

with periodic boundary conditions where (with a slight abuse of notation) we used f instead of f_{ext} to denote the source term. The source term f may have discontinuities in the domain \mathbf{B} . 1-D results for the adaptive solution for the case $d = 1$ were reported in [1, 2, 3, 10].

Let us consider the projection L_0 of the periodized operator Δ on \mathbf{V}_0 , the finest scale under consideration, $L_0 = P_0 \Delta P_0$ and L_s and L_{ns} are its standard (s-form) and non-standard (ns-form) forms (representations) of the operator \mathcal{L} in the wavelet basis, respectively. The s- and ns-forms are well described in [1, 8].

One of the difficulties in solving (3.2) stems from the inherently large condition number of the linear system resulting from the discretization of (3.2). As it was shown in [1, 9, 7], using a diagonal preconditioner in the wavelet system of coordinates yields a linear system with the condition number typically less than 10, independently of its size. Let \mathcal{P} denote such a diagonal preconditioner.

In [7], the s-form is used to solve the two-point boundary value problem. Alternatively, we may use the ns-form. Some care is required at this point since the preconditioned ns-form is dense unlike the s-form, which remains sparse. Thus, in the process of solving the linear system, it is necessary to apply the preconditioner and the ns-form sequentially in order to maintain sparsity. The ns-form is preferable in multiple dimensions since, for example, differential operators require $O(1)$ elements for representation on all scales (see e.g. [9]).

We develop a constrained preconditioned CG algorithm for solving (3.2) in an adaptive manner ([1]). Both the s-form and the ns-form may be used for this purpose but it appears that using the ns-form is more efficient especially if compactly supported wavelets are used and high accuracy is required.

Let us consider (3.2) in the wavelet system of coordinates

$$L_{ns}u_w = f_w, \quad (3.4)$$

where f_w and u_w are representations of f and u in the wavelet system of coordinates. This equation should be understood to include the rules for applying the ns-form (see [8]).

Let us rewrite (3.4) using the preconditioner \mathcal{P} as

$$\mathcal{P} L_{ns} \mathcal{P} v = \mathcal{P} f_w, \quad (3.5)$$

where $\mathcal{P} v = u$. For example, for the second derivative the preconditioner \mathcal{P} is as follows:

$$\mathcal{P}_{il} = \delta_{il} 2^j \quad (3.6)$$

where $1 \leq j \leq n$ is chosen depending on i, l so that $n - n/2^{j-1} + 1 \leq i, l \leq n - n/2^j$, and $\mathcal{P}_{nn} = 2^n$.

The periodized operator Δ has the null space of dimension one which contains constants. If we use the full decomposition (over all n scales) in the construction of the ns-form then the null space coincides with the subspace \mathbf{V}_n which in this case has dimension one (see [7]). This allows us to solve (3.5) on the range of the operator $\bigoplus_{1 \leq j \leq n} \mathbf{W}_j$ where the linear system (3.5) is well conditioned.

3.1.1 The solution strategy

Let us represent the source term f and the solution u in (3.3) in the wavelet basis,

$$f(x) = \sum_{j \leq n} \sum_{\mathbf{k}} \sum_{\sigma} f_{j,\mathbf{k}}^{\sigma} \psi_{j,\mathbf{k}}^{\sigma}(x) + \sum_{\mathbf{k}} s_{n,\mathbf{k}}^f \varphi_{n,\mathbf{k}}(x), \quad (3.7)$$

$$u(x) = \sum_{j \leq n} \sum_{\mathbf{k}} \sum_{\sigma} u_{j,\mathbf{k}}^{\sigma} \psi_{j,\mathbf{k}}^{\sigma}(x) + \sum_{\mathbf{k}} s_{n,\mathbf{k}}^u \varphi_{n,\mathbf{k}}(x), \quad (3.8)$$

where

$$f_{j,\mathbf{k}}^{\sigma} = \langle f, \psi_{j,\mathbf{k}}^{\sigma} \rangle, \quad u_{j,\mathbf{k}}^{\sigma} = \langle u, \psi_{j,\mathbf{k}}^{\sigma} \rangle, \quad s_{n,\mathbf{k}}^f = \langle f, \varphi_{n,\mathbf{k}} \rangle \quad \text{and} \quad s_{n,\mathbf{k}}^u = \langle u, \varphi_{n,\mathbf{k}} \rangle. \quad (3.9)$$

We now define the ϵ -accuracy subspace for f to be the subspace on which f may be represented with accuracy ϵ , namely,

$$M_{r.h.s}^{\epsilon} = \mathbf{V}_n \bigcup \{ \text{span}\{\psi_{j,\mathbf{k}}^{\sigma}\} \mid (j, \mathbf{k}, \sigma) : |f_{j,\mathbf{k}}^{\sigma}| > \epsilon \}, \quad (3.10)$$

and observe that the ϵ -accuracy subspace for the solution

$$M_{sol}^{\epsilon} = \mathbf{V}_n \bigcup \{ \text{span}\{\psi_{j,\mathbf{k}}^{\sigma}\} \mid (j, \mathbf{k}, \sigma) : |u_{j,\mathbf{k}}^{\sigma}| > \epsilon \} \quad (3.11)$$

may be estimated given $M_{r.h.s}^{\epsilon}$.

The ϵ -accuracy subspace for f in the wavelet multiresolution expansion, which is the indices of the wavelet coefficients above the threshold ϵ , are called masks and denoted by M^{ϵ} .

There are three main features in our approach to solve (3.3) :

1. **Estimation of the ϵ -accuracy subspace for the solution.** Our first step is to explicitly estimate the subspace M_{sol}^ϵ given $M_{r.h.s}^\epsilon$. For elliptic operators the dimension of M_{sol}^ϵ is proportional to that of $M_{r.h.s}^\epsilon$.
2. **Preconditioning of the operator.** A simple diagonal preconditioner is available for periodized differential operators in wavelet bases [9, 7] which yields a condition number that is of $O(1)$. We will show in Section 3.3 how to construct simple preconditioner in wavelet bases for more general operators.
3. **Constrained Iterative Solver.** We use preconditioned Conjugate Gradient (CG) method which we constrain to the subspace estimated at Step 1, e.g. $M_{\lambda,\mu}$ (this subspace is described in 3.13. The CG method requires only a constant number of iterations due to preconditioning at Step 2, whereas the cost of each iteration is proportional to the dimension of M_{sol}^ϵ provided we succeed to limit the number of operations required for the application of the operator (matrix) in the CG method (see below).

Steps 1-3 constitute an adaptive algorithm for solving Poisson's equation. The general flow of the algorithm is:

Initialization

$$R \leftarrow f$$

Main loop

$$M_{r.h.s}^\epsilon = \text{big coefficients (above threshold } \epsilon) \text{ of } R$$

Solve loop

$$\Delta u = R \text{ on } M_{r.h.s}^\epsilon$$

$$R \leftarrow R - \Delta u$$

The **solve loop** contains two computations:

1. PCG is applied to $P\Delta P^T u$, where P is a preconditioner
2. The P and $P^T u$ are applied on the standard form while Δ is described by the non-standard form.

Proposition 3.1 ([1]) *Let*

$$u(x) = \sum_j \sum_{\mathbf{k}} \sum_{\sigma} u_{j,\mathbf{k}}^\sigma \psi_{j,\mathbf{k}}^\sigma(x) + \text{constant} \quad (3.12)$$

be the solution of

$$\Delta u = \psi_{j',\mathbf{k}'}^{\sigma'} \quad x \in \mathbf{B} \quad (3.13)$$

with periodic boundary conditions. For any $\epsilon > 0$ there exist $\lambda > 0$ and $\mu > 0$ such that all indices (j, \mathbf{k}, σ) corresponding to the significant coefficients of the solution, $|u_{j,\mathbf{k}}^\sigma| \geq \epsilon$, satisfy $|\mathbf{k} - \mathbf{k}'| \leq \lambda$ and $|j - j'| \leq \mu$.

The size of $\mu > 0$ and $\lambda > 0$ depends on the particular choice of basis and, of course, on ϵ . Given $M_{r.h.s}^\epsilon$, we may construct the set $M_{\lambda,\mu}$ as a (λ, μ) -neighborhood of $M_{r.h.s}^\epsilon$. According to Proposition 3.1, $M_{sol}^\epsilon \subset M_{\lambda,\mu}$. We note that estimating the subspace amounts to constructing a mask, which contains indices of significant coefficients. Instead of estimating $M_{\lambda,\mu}$ directly, we may use an iterative approach [1].

3.1.2 The relation between the masks on the d and s coefficients

The solution strategy requires that after having multiscale decomposition of the r.h.s we create masks on the d coefficients. The masks on the d coefficients are determined according to a predefined threshold (accuracy) ϵ . This way we sparsify all the d scales in the multiscale. But during the application of the multiscale wavelet decomposition/reconstruction the s part of the multiscale is filled up (it becomes “dense”). This causes us to loose the sparsity advantages we gain on the d coefficients. It is proved in [1], that although the s_k^j coefficients are generally dense it suffices to consider in the non-standard form only those labels (j, k) near the corresponding labels of d_k^j that are used to define the mask. In other words, the same sparsity that was achieved on the d coefficients can be imposed upon the s coefficients. This way the whole process of the multiscale decomposition/reconstruction during the solution is kept sparse. Also masking reduces the number of coefficients that should be used during sparse multiplication and removes speed-up degradation that exists in original multiplication algorithm while using all levels during multiplication.

3.2 Iterative Solver

In order to solve (3.5) we apply the Conjugate Gradient method constrained to the subspace $M_{\lambda,\mu}$. Without such constraint the conjugate directions become “dense” at early stages of the iteration only to become small outside the subspace $M_{\lambda,\mu}$ later. Thus, constraining the solution to a subspace is critical for an adaptive algorithm.

In applying the conjugate gradient method in the wavelet coordinates, we generate only those entries of conjugate directions which are in the set of significant indices which define the subspace $M_{\lambda,\mu}$ (called the masks). This yields an algorithm where the number of operations at each iteration is proportional to the number of elements of $M_{\lambda,\mu}$. The number of iterations is $O(1)$ and, thus, the overall number

of operations is proportional to the number of significant coefficients of f , i.e., the dimension of $M_{r.h.s}^\epsilon$.

3.2.1 Operators with variable coefficients.

As in the case of the Laplacian, the ϵ -accuracy subspace for the solution may be estimated using corresponding subspaces for the r.h.s and the coefficients. Essentially, we consider the union of such subspaces as a starting point for constructing $M_{\lambda,\mu}$. These estimates may be revised in the process of iteration. The proof, which the above algorithm is applicable to operators with variable coefficients, is given in [1].

3.3 Preconditioner

Let us demonstrate how to construct a diagonal preconditioner for the sum of operators $-\Delta + Const$ in wavelet bases. We observe that if A and B are diagonal operators with diagonal entries a_i and b_i , where $a_i + b_i \neq 0$, then the diagonal operator with entries $1/(a_i + b_i)$ is an ideal preconditioner for operator $A + B$.

In our case, the operator $-\Delta$ is not diagonal, but it has a good diagonal preconditioner, Eq. (3.6), in wavelet bases (3.6), see [1, 2]. Let us use this preconditioner instead of $-\Delta$ for the purpose of constructing a preconditioner for $-\Delta + Const$, where $Const > 0$. We note that in wavelet bases the identity operator remains unchanged. We restrict $Const \cdot I$, where I is the identity operator, to the subspace $\bigoplus_{1 \leq j \leq n} \mathbf{W}_j$ and construct a preconditioner on this subspace. We obtain

$$\mathcal{P}_{il} = \frac{\delta_{il}}{\sqrt{2^{-2j} + Const}} \quad (3.14)$$

where $1 \leq j \leq n$ is chosen depending on i, l so that $n - n/2^{j-1} + 1 \leq i, l \leq n - n/2^j$, and $\mathcal{P}_{nn} = 1/\sqrt{2^{-2n} + Const}$. The square root appears in (3.14) in order to symmetrize the application of the preconditioner as shown before.

It is important to notice that mentioned above preconditioner is suitable for operator $-\Delta + Const$, while we actually solving problem with non-constant coefficients $-\Delta + b(x)$, that in general could have another preconditioner. But in our numerical experiments we still used the constant-based preconditioner of the form 3.14, however we had to choose manually the constant of the preconditioner. We do not have yet strict mathematical dependence between $b(x)$ and constant in Eq. 3.14. This is left for a future research. We made numerical comparison between different constants of preconditioner and results are shown in Fig. 3.6.

3.4 PDE Solver: 1-D pseudo code

We are describing the solver for the equation like:

$$\frac{d^2u}{dx^2} - b(x)u(x) = f(x). \quad (3.15)$$

It is an extension of the algorithm that was described in [1, 2, 3] to non-homogeneous equations of the form 3.15. Assume that the r.h.s has size $N = 2^M$, so we can decompose it into M levels. The basic 1-D structure in the algorithm consists of an array of pointers of size $2M + 1$. Each odd entry in this array, $2j - 1$, $j = 1, \dots, M$, is a pointer to a sparse vector of size $N/2^j$ where we store the “s” (averaged) wavelet coefficients of scale j . Each even entry in this array, $2j$, $j = 1, \dots, M$, is a pointer to a sparse vector of size $N/2^j$ where we store the “d” (derivative, high-pass) wavelet coefficients of scale j . The 1-D algorithm uses only this basic structure for $w_t, w_r, w_p, w_{p_1}, w_s, w_x, w_b$, which are used in the pseudo code.

1. Preprocessing -

Second derivative in the NSF form - computes the non-standard form of the second derivative according to the filter type and filter size, number of scales in the multiresolution decomposition, and the required accuracy. The second derivative operator is computed as explained in [1].

Setup of the r.h.s.-

1. Find the wavelet coefficients of the exact solution u by multiresolution decomposition of u .
2. Transform the coefficients of the exact solution to a sparse data structure by thresholding the wavelet coefficients.
3. Find the wavelet coefficients of the b by multiresolution decomposition of b .
4. Transform the coefficients of the b to a sparse data structure w_b by thresholding the wavelet coefficients.
5. Reconstruct of the r.h.s. into a sparse data structure.
6. Apply the second derivative operator on exact solution which is multiplication of the vector by ns-form, and add to result multiplication of exact solution on b .

Wavelet decomposition of the r.h.s - store it in w_t

Wavelet decomposition of the v - store it in w_v

Mask on the “s” coefficients - this mask is determined by the “s” wavelet coefficients w_t of the decomposed r.h.s by thresholding.

Apply the preconditioner - on the wavelet coefficients w_t of the r.h.s.

Copy - $w_t \rightarrow w_r$

Compute the L_2 norm of w_r - denoted by ρ

2. The main iteration loop of the CG -

If first iteration -

$$w_r \rightarrow w_p$$

$$w_r \rightarrow w_{p_1}$$

If not the first iteration - $\beta = \frac{\rho}{\rho_1}$
sparse linear combination: $w_r + \beta w_p \rightarrow w_{p_1}$.

$$w_{p_1} \rightarrow w_p$$

Apply preconditioner - on w_p

Sparse wavelet reconstruction - from w_p using the masks.

Apply the left hand side on w_p using the masks (by sum of application of non-standard form and sparse multiplication by b) to get w_s .

Apply preconditioner - on w_s

Sparse inner product - of w_s and w_{p_1} to produce r .

Compute - $\alpha = \frac{\rho}{r}$

Sparse linear combination - $w_x + \alpha w_{p_1} \rightarrow w_q$.

Copy - $w_q \rightarrow w_x$.

Sparse linear combination - $w_r - \alpha w_s \rightarrow w_q$.

Copy - $w_q \rightarrow w_r$

Save - $\rho_1 = \rho$

Copy - $w_{p_1} \rightarrow w_p$

Go to beginning of the loop (2)

3.4.1 Estimation of the number of operations

Equation 3.15 is solved using the Conjugate Gradient method ([16]), where the computation of the inverse operator is not necessary (just the application of the operator itself is required). This property enables us to use these methods for solving problems where the operator's kernels are separable.

All the computations of the solver are performed within the masks as was explained in [1, 2, 3, 8]. The wavelet transform of the solution consists of N_s significant coefficients in the masks which are concentrated near the singularities (see [1]).

Although N is the total number of discretization points in each direction, we will use for our estimations the size of N_s .

Usually, the convolution with filter of size l takes $2l$ operations. By utilizing factorization properties of the wavelet filters, as was described in [5, 6], we can reduce the number of operations of the 1-D convolution by a factor of at least 2.

During each iteration of the CG, the kernel is decomposed into the standard form (for the application of the preconditioner and reconstruction back) and then the non-standard form is applied once. We assume that $2lN_s$ operations are needed for 1-D sparse wavelet convolution with filter of length l . The same is true for the application of the non-standard form. Application of sparse multiplication with parameters j_f , J , and j_0 , takes $O(2^M - j_f - j_0)$ operations. Sparse inner product needs $2N_s$ operations. And the same is true for sparse linear combination. Therefore, one iteration of the solver requires $4lN_s + 3 \cdot 2N_s + O(2^M - j_f - j_0) = (4l + 6)N_s O(2^M - j_f - j_0)$ operations. Therefore, the total number of operations for N_{iter} is $N_{iter} \cdot (4l + 6)N_s + O(2^M - j_f - j_0)$. To get ϵ -accuracy we need $(N_{iter} \cdot (4l + 6)N_s + O(2^M - j_f - j_0)) \log \epsilon$ operations.

The algorithm is very efficient as compared to the dense multiplication method which will require $(N_{iter} \cdot (4l + 6)N_s + O(N)) \log \epsilon$ operations. The efficiency of the process relies on the fact that we can take appropriate parameters of sparse multiplication having on one side the necessary accuracy of the multiplication and from the other side maintains $2^M - j_f - j_0 \ll N$ obtaining good speed up - see section 3.5.

3.5 Numerical results

In this section, we discuss the influence of the algorithm's parameters on the run time and on the accuracy of the equation's solver. Let us analyze how the PDE's solver performs using adaptive multiplication comparing with regular ("dense") multiplication.

3.5.1 Example 1: Smooth function

Assume we have

$$\frac{d^2u(x)}{dx^2} - b(x)u(x) = f(x) \quad (3.16)$$

where the r.h.s $f(x)$ is numerically computed with the wavelet derivative operator such that the exact solution u and b are:

$$\begin{aligned} u(x) &= e^{-256.0 \cdot x^2} \\ b(x) &= .03 + .017 \cdot e^{-50 \cdot (x-.1) \cdot x}. \end{aligned} \quad (3.17)$$

We can see in Fig. 3.5 how the accuracy and the run time are affected by different parameters of the multiplication algorithm.

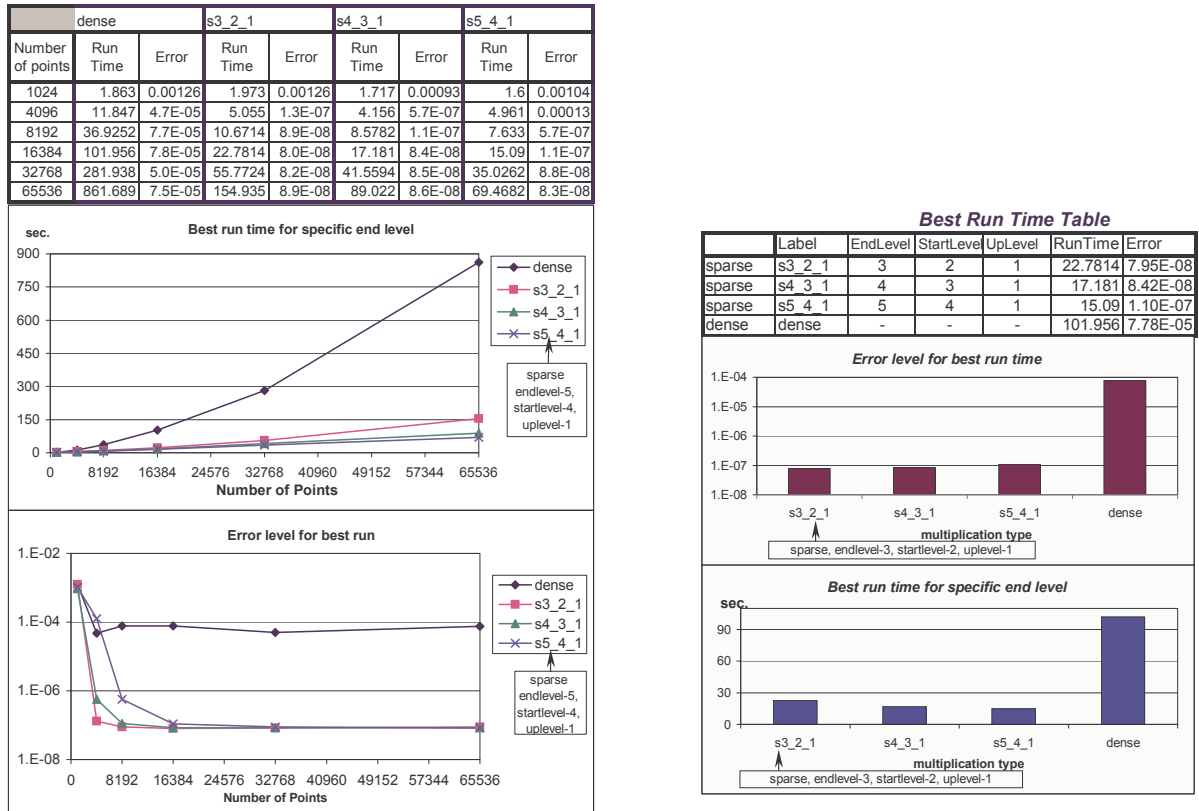


Figure 3.5: Dependency of the PDE solver's run time and accuracy on: Left N . Right: J and j_f

3.5.2 Example 2: Non-smooth function

We use Eq. (3.16) where the r.h.s $f(x)$ is numerically computed with the wavelet derivative operator such that the exact solution u and b are:

$$\begin{aligned}
 i &= 0 \dots N - 1 \\
 x &= -.5 + i/(N - 1) \\
 u(x) &= x \cdot e^{-256.0 \cdot x^2} + 20.0 \cdot (0.2 + x) \cdot e^{-32000.0 \cdot (0.2+x)^2} \\
 b(x) &= .03 + .017 \cdot e^{-50 \cdot (x-.1) \cdot x}
 \end{aligned} \tag{3.18}$$

where $u(x)$ is seen in Fig. 2.2.

The dependency of the accuracy on the preconditioner constant (Eq. 3.14) is shown in Fig. 3.6. To generate this graph, we calculate the solution of Eq. 3.18 on 8192 data points and used dense multiplication. In Fig. ??, we can see how the accuracy and the run time are affected by different parameters of multiplication algorithm.

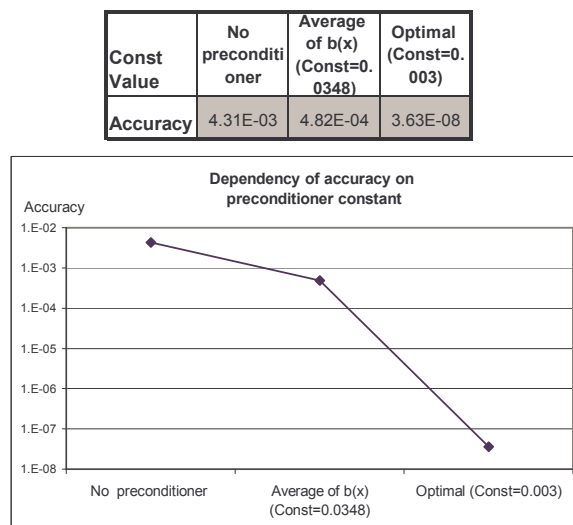


Figure 3.6: Dependency of the PDE solver accuracy on the preconditioner constant.

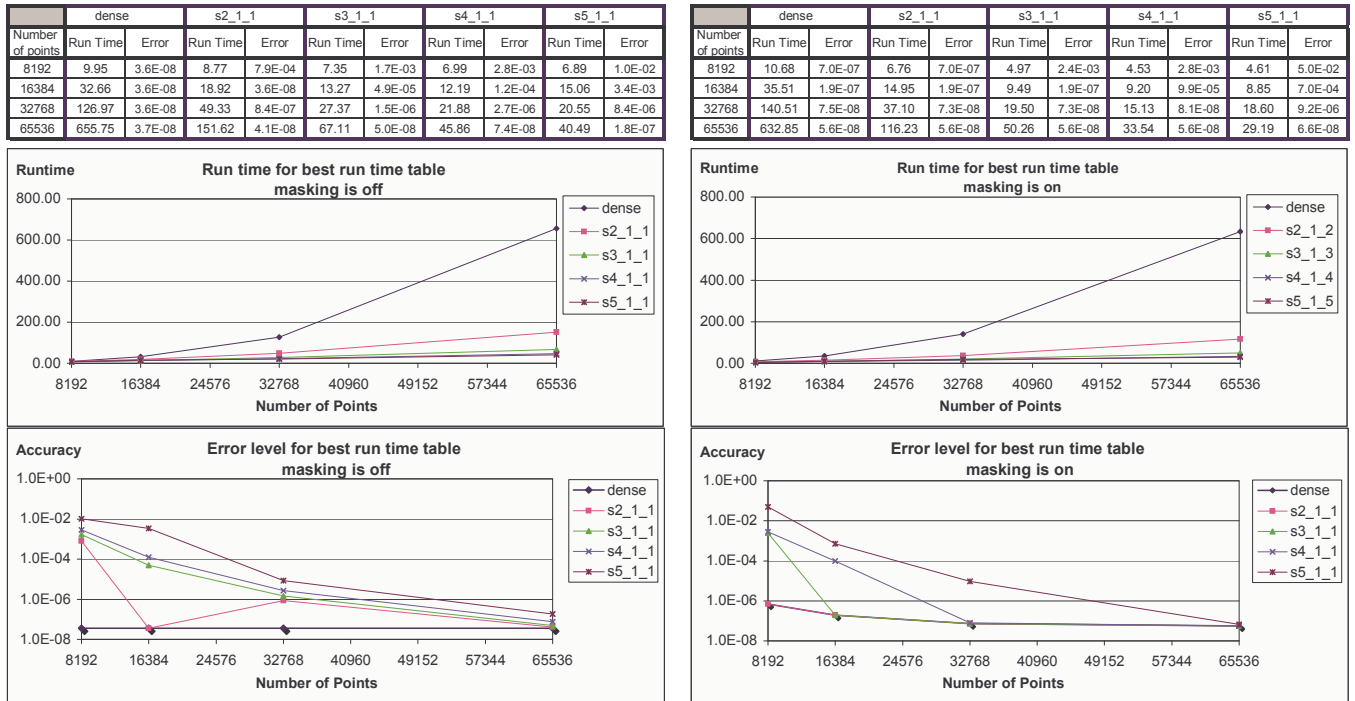


Figure 3.7: The PDE’s solver run time and accuracy, Left: without masking. Right: with masking

As we see, the run time is dramatically improved if the adaptive multiplication is used with the masking procedure. For example, the run time for 65536 points can be lowered more than 20 times with less than one third accuracy loss. The best speedup was more than 20 times faster (29 against 632 sec.) and it was obtained using $j_f = 1$, $J = 5$ and $j_0 = 1$ with 17% accuracy loss (if comparing it against a run with dense multiplication and s-masking - see Fig. ??). Our implementation was most efficient on large data. The total iterations time was reduced from 635 to 29 seconds. To verify the run time, we executed the PDE’s solver five times with the same parameters. Average time was calculated and it is presented.

4 Conclusion

The algorithm, which is described in this paper, uses adaptive multiplication that allows dramatic improvement of the run time of a 1-D PDE solver. The original algorithm in [1] was improved. The dependency of the multiplication parameters on the achieved run time and accuracy of the solver is investigated. The numerical

results show that a speed-up of up to a factor of 20 can be achieved using our implementation of the adaptive multiplication. The advantage of adaptive multiplication can be demonstrated when solving PDEs on large datasets.

We do not have yet strict mathematical dependence between $b(x)$ and the constant in Eq. 3.14. Therefore, this is left for a future research. In addition, extension of the adaptive multiplication to higher dimensional problems is a promising direction.

References

- [1] A. Averbuch , G. Beylkin , R. Coifman , P. Fischer , M. Israeli, *Adaptive Solution of Multidimensional PDEs via Tensor Product Wavelet Decomposition*, International Journal of Pure and Applied Mathematics, 44(1), 75-115, 2008.
- [2] A. Averbuch, G. Beylkin, R. Coifman, M. Israeli, *Multiresolution Solution of Elliptic and Parabolic PDEs*, The Samuel Neaman Workshop on Signal and Image Representation in Combined Space. Technon, Haifa, Israel, May 8-11,1994: edited by Y. Zeevi, R. Coifman, pp.341-360, Academic Press, 1998
- [3] A. Averbuch, G. Beylkin, R. Coifman, P. Fischer, M. Israeli, *A Wavelet Based Constrained Preconditioned Conjugate Gradient for Elliptic Problems*, Conference on Preconditioned Iterative Solutions Methods for Large Scale Problems in Scientific Computations, May 27-29,1997 University of Nijmegen, The Netherlands.
- [4] Averbuch, A., Beylkin, G., Coifman, R., Fischer P., Israeli, M., *A wavelets based constrained Preconditioned Conjugate Gradient for elliptic problems*, Conference on Preconditioned Iterative Solution Methods for Large Scale Problems in Scientific Computations, May 27-29, 1997 University of Nijmegen, The Netherlands.
- [5] A. Averbuch, M. Israeli, F. Meyer, *Speed vs. Quality in Low Bit-Rate Still Image Compression*, Signal Processing: Image Communication, 15, 231-254, 1999.
- [6] A. Averbuch, F. Meyer, J-O Strömberg, *Fast Adaptive Wavelet Packet Image Compression*, IEEE Trans. on Image Processing, 9:5, 792-800, 2000..
- [7] Beylkin, G., *On wavelet-based algorithms for solving differential equations*, In John J. Benedetto and Michael W. Frazier, editors, *Wavelets: Mathematics and Applications*, pages 449–466. CRC Press, 1994.

- [8] Beylkin, G., Coifman, R., Rokhlin, V., *Fast Wavelet Transforms and Numerical Algorithms I*, Comm. on Pure and Applied Mathematics, Vol. XLIV, pp. 141-183, 1991.
- [9] Beylkin, G., *On the Representation of Operators in Bases of Compactly Supported Wavelet Bases*, SIAM J. Num. Anal., Vol. 6, pp. 1716-1740, 1992.
- [10] G. Beylkin, J.M. Keiser *An Adaptive Pseudo-Wavelet Approach for Solving Non-linear Partial Differential Equations*
- [11] J.M. Bony, *Calcul symbolique et propagation des singularit'es pour les 'equations aux d'eriv'ees partielles non-lin'eaies*. Ann. Scient. E.N.S. 14 (1981), 209.
- [12] A. Cohen, W. Dahmen, R. DeVore, *Adaptive Wavelet Methods for Elliptic Operator Equations Convergence Rates*, October 12 1998.
- [13] A. Cohen, R. Mason, *Wavelets Method for Second Order Elliptic Problems, Preconditioning and Adaptivity*.
- [14] A. Cohen, R. Mason, *Adaptive Wavelet Method for Second Order Elliptic Problems. Boundary Conditions and Domain Decomposition*.
- [15] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, 1992.
- [16] Golub, G.H., Van Loan, C.H., *Matrix Computations*, (J. Hopkins University Press, Baltimore), 1983.
- [17] S.G. Mallat, *Multiresolution Approximations and Wavelet Orthonormal Bases for $L^2(\mathbb{R})$* , Trans. of AMS, Sept. 1989.
- [18] S.G. Mallat, *A wavelet tour of signal processing*, Academic Press, Second Edition, 1999.