Multiscale Computation with Interpolating Wavelets

Ross A. Lippert*, T.A. Arias†, and Alan Edelman‡

In press; Submitted Nov. 12, 1996

Classifications: 65D05, 65D30, 82A71, 81C06, 81C07

Keywords: lda, wavelets, 'interpolating wavelet', 'interpolating scaling function', interpolets, 'electronic structure', 'multiscale computation', 'nonstandard multiplication', 'autocorrelation function'

Computational support provided by the MIT Xolas project and general support provided by the State Street Bank Science Partnership Fund at MIT

*Department of Mathematics Room 2-342, Massachusetts Institute of Technology, Cambridge, MA 02139, ripper@mit.edu, supported by a grant from the Liberty Mutual Group Fund for scholarships and financial aid for graduate mathematics students.

†Department of Physics Room 12-110, Massachusetts Institute of Technology, Cambridge, MA 02139, muchomas@mit.edu, supported in part by the MRSEC Program of the National Science Foundation under award number DMR 94-00334, the Alfred P. Sloan Foundation under award number BR-3456.

‡Department of Mathematics Room 2-380, Massachusetts Institute of Technology, Cambridge, MA 02139, edelman@math.mit.edu, supported by a fellowship from NSF Grant 9404326-CCR.

Multiscale Computation with Interpolating Wavelets

Ross A. Lippert
905 Main St. #10
Cambridge, MA
O2139
ripper@mit.edu
(617) 661-2879

Abstract

Multiresolution analyses based upon interpolets, interpolating scaling functions introduced by Deslauriers and Dubuc, are particularly well-suited to physical applications because they allow *exact* recovery of the multiresolution representation of a function from its sample values on a *finite* set of points in space. We present a detailed study of the application of wavelet concepts to physical problems expressed in such bases. The manuscript describes algorithms for the associated transforms which, for properly constructed grids of variable resolution, compute correctly without having to introduce extra grid points. We demonstrate that for the application of local homogeneous operators in such bases, the non-standard multiply of Beylkin, Coifman and Rokhlin[4] also proceeds exactly for inhomogeneous grids of appropriate form. To obtain less stringent conditions on the grids, we generalize the non-standard multiply so that communication may proceed between non-adjacent levels. The manuscript concludes with timing comparisons against naïve algorithms and an illustration of the scale-independence of the convergence rate of the conjugate gradient solution of Poisson's equation using a simple preconditioning.

1 Introduction and Motivation

Wavelets offer a means of approximating functions that allows selective refinement. If regions of an image or a signal have exceptionally large variations, one need only store a set of coefficients, determined by function values in neighborhoods of those regions, in order to reproduce these variations accurately. In this way, one can have approximations of functions in terms of a basis that has spatially varying resolution. This approach reduces the memory storage required to represent functions and may be used for data compression.

Physical applications often involve multiple physical fields which interact in space with non-linear couplings. Efficient implementations must minimize not only storage to represent the these fields but also the processing required to describe their interactions. It is highly desirable to perform the needed operations with a fixed, limited number of floating point operations for each expansion coefficient and to minimize the number of points in space at which physical interactions must be evaluated.

As a concrete example of a realistic application, consider the computation of the quantum mechanical electronic structure of a collection of atoms in three dimensions. For other examples of physical applications, the reader may wish to consult [3], [5], [17], *et al.*. Arias and coworkers [9],[2], and other works which have appeared after the original submission of this manuscript nearly one year ago [27],[26], have studied the use of multiresolution bases in quantum mechanical computations. (For a review, see [1].) It is a consequence of quantum physics that, near atomic nuclei, electronic wave functions vary rapidly and, that far from the nuclei, the wave functions tend to be much more smooth. From this observation, one can anticipate that the fine scale coefficients in a multiresolution analysis of the electronic wave functions and associated physical fields will be significant only near the atomic cores, allowing for truncation. This problem is thus an ideal candidate for multiresolution techniques.

Within density functional theory [19], the quantum physics of electrons and nuclei involves two types of fields, the Schrödinger wave function $\{\psi_i(r)\}$ for each electron i and the electrostatic potential $\phi(r)$ arising from the average electronic density $n(r) \equiv \sum_i |\psi_i(r)|^2$. Within the local density approximation (LDA) [20], the solution for the correct values of these fields is obtained at the saddle-point of lowest energy of the Lagrangian functional

$$\begin{aligned} \mathcal{L}_{LDA}(\{\psi_i\}, \phi) &= \frac{1}{2}\sum_i f \int d^3r \, ||\nabla\psi_i(r)||^2 + \int d^3r \, V_{\text{nuc}}(r) n(r) + \int d^3r \; \epsilon_{xc}(n(r))n(r) \\ &- \int d^3r \, \phi(r)n(r) - \frac{1}{8\pi}\int d^3r \, ||\nabla\phi(r)||^2. \end{aligned} \tag{1}$$

Here, we work in units $\hbar = m = e = 1$, $V_{\text{nuc}}(r)$ is the total potential which each electron feels due to the presence of the atomic nuclei, and $\epsilon_{xc}(n)$ is a non-algebraic function known only through tabulated values. (For a review of density functional theory, see [23].)

In practice, one finds the fields $\{\psi_i(r)\}$, $\phi(r)$ by

- expanding the fields in terms of unknown coefficients within some basis set

$$\begin{aligned} \psi_i(x) &= \sum_\alpha c_{\alpha,i} b_\alpha(x) \\ \phi(x) &= \sum_\alpha d_\alpha b_\alpha(x); \end{aligned} \tag{2}$$

- evaluating Eq. (1) in terms of the unknown coefficients c and d;
- determining analytically the gradients of the resulting $\mathcal{L}_{LDA}(c, d)$ with respect to those coefficients; and
- proceeding with conjugate gradients to locate the saddle point.

All follows directly once one has expressed the Lagrangian as a function of the expansion coefficients.

In doing this, we note that each term represents a local coupling in space, but that one coupling, $\phi(r)n(r)$, is cubic in the field coefficients c and d, and another, $\epsilon_{xc}(n(r))n(r)$, is only known in terms of tabulated values. Expanding the product of two wavelets in terms of wavelets on finer levels would make possible the treatment of the cubic coupling to some level of approximation. (See, for example, [6].) However, this route becomes increasingly difficult for higher order interactions and is hopeless for non-algebraic or tabulated interactions, such as $\epsilon_{xc}(n(r))$. For higher order interactions it is natural, and for non-algebraic and tabulated interactions necessary, to evaluate the interactions at some set of points in space and then recover expansion coefficients for the result. One then relies upon the basis set to provide interpolation for the behavior at non-sample points.

The benefits of both truncated wavelet bases and interpolation on dyadically refined grids are given by the use of interpolating scaling functions [16], [7], [8], [14], [15] (or interpolets [28], [1]), which are functions with the following properties (from [15], pp. 6-7).

Let $\phi(x)$ be an interpolet, then

(INT1) **cardinality:** $\phi(k) = \delta_{0,k}$ for all $k \in Z^n$

(INT2) **two-scale relation:** $\phi(x/2) = \sum_{y \in Z^n} c_y \phi(x - y)$

(INT3) **polynomial span:** For some integer $m \geq 0$, any polynomial $p(x)$ of degree m can be represented as a formal sum $\sum_{y \in Z^n} a(y)\phi(x-y)$.

Cardinality allows the fast conversion between uniform samples and interpolating scaling functions and has subtle yet profound consequences for the resulting multiresolution basis. In particular, as is evident from our algorithms below, the expansion coefficient for a basis function on a particular scale is independent of the samples of the function for points associated with finer scales. Consequently, the expansion coefficients which we obtain for functions maintained in our basis are identical to what would be obtained were function samples available on a complete grid of *arbitrarily* fine resolution. This eliminates all error in the evaluation of non-linear, non-algebraic and tabulated interactions beyond the expansion of the result in terms of a finite set of basis functions.

The c_y in the two-scale relation are referred to as scaling coefficients, and cardinality actually implies that $c_y = \phi(y/2)$. The two-scale relation allows the resolution to vary locally in a mathematically consistent manner.

The polynomial span condition captures, in some sense, the accuracy of the approximation. By cardinality, we actually have $a(y) = p(y)$. We shall call m the polynomial order.

Interpolets thus can be thought of as a bridge between computations with samples on dyadically refined grids and computations in a multiresolution analysis. The former point of view is useful for performing local nonlinear operations, while the latter is useful for the application of local linear operators.

This manuscript explores $O(N)$ algorithms that calculate transforms and linear operators for grids of variable resolution but return, for the coefficients considered, *exactly* the same results as would be obtained using a full, uniform grid at the highest resolution *without* the need to introduce artificial temporary augmentation points to the grid during processing. We thus show that with relatively mild conditions on the variability of the resolution provided by the grid, interpolet bases provided the ultimate economy in the introduction of grid points: only as many samples in space need be considered as functions used in the basis. The four transforms (forward, inverse, and the dual to each) mapping between coefficients and functions samples which we discuss here are particular to interpolet bases. For the application of operators in such bases, we show that the familiar non-standard multiply of Beylkin, Coifman and Rokhlin[4] shares with the transforms the property of correctness without the need to introduce additional grid points. Furthermore, we weaken the condition on grid variability by using a modification of the non-standard multiply. We generalize the non-standard multiply so that communication may proceed between nearby but non-adjacent levels and thereby obtain less stringent conditions on the variability of the grid. All of theoretical results in this manuscript are presented in a general d-dimensional space. Illustrative examples for the purpose of discussion will be given in $d = 1$ and $d = 2$ dimensions. The examples of applications in the final section will be in $d = 3$ dimensions. Our focus is entirely on interpolet bases, and so it remains an open question whether these results hold true or can be adapted to more general wavelet systems.

Our organization is as follows. In Sections 2 and 3, we explain how to manipulate and construct interpolet expansions and some aspects of how well they perform. These sections will present nothing new to the experienced wavelet user, but will explain our notational conventions and recapitulates common theorems ([12], [24], [13], [25], et al.) for wavelet novices. In Section 4, we describe how nonuniform bases can be conceptualized in the framework of interpolet expansions and then use our results to develop algorithms for the transforms. Section 5 details the algorithm for ∇^2 and other operators. Section 6 gives some practical details for the reader interested in implementing these algorithms. Finally, Section 7 compares, in three dimensions, timings of these implementations with the timings of naïve algorithms. This final section also explores the convergence of a preconditioned conjugate gradients algorithm in the solution of Poisson's equation for the full three dimensional electrostatic potential arising from the nuclei in the nitrogen molecule.

2 Introduction to Interpolets

There is a unique set of interpolets on R having symmetry and minimal support for a given polynomial order $m = 2l - 1$ (the *Deslauriers-Dubuc functions* [14]). These are the functions with which this article is primarily concerned (our results carry over to more general interpolets, and no use will actually be made of minimal support or symmetry).

To determine the c_y's, one sets $c_{2j} = \delta_{m0}$ and $c_y = c_{-y}$. One may solve the Vandermonde system,

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 3^2 & \cdots & (2l-1)^2 \\ 1 & 3^4 & \cdots & (2l-1)^4 \\ & & \cdots & \\ 1 & 3^{2l-2} & \cdots & (2l-1)^{2l-2} \end{pmatrix} \begin{pmatrix} c_1 \\ c_3 \\ . \\ . \\ c_{2l-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ . \\ . \\ 0 \end{pmatrix}$$

to obtain the remaining c_y's. These coefficients satisfy the conditions for polynomial order $2l - 1$.

The scaling coefficients for $m = 1$ are

$$c_{-1} = c_1 = 0.5, c_0 = 1,$$

and for $m = 3$ (the example used for Figure 1.) they are

$$c_{-3} = c_3 = -\frac{1}{16}, c_{-1} = c_1 = \frac{9}{16}, c_0 = 1.$$

One may then take tensor products of ϕ's and c_y's to form interpolets in higher dimensions.

2.1 Interpolet Multiresolution Analysis

We are concerned with recursive representations of functions from samples at integer points on both uniform and refined grids. There are many definitions which make the exposition more clear.

Definition 2.1 *For $k > 0$, let $C_k = 2^k Z^n$, and let $D_k = C_{k-1} - C_k$. For $k \leq 0$, let $C_k = Z^n$, and $D_k = \emptyset$.*

We consider C_k to be the set of coarse lattice points on the lattice $2^{k-1}Z^n$ and D_k, the detail lattice points, to be those points on $2^{k-1}Z^n$ which are not coarse. Note: $D_k \cup C_k = C_{k-1}$, and $Z^n = C_k \cup D_k \cup D_{k-1} \cup \cdots \cup D_1$ is a partition of Z^n.

Definition 2.2 *We let $\theta_k(y) = min(k, m)$ where m is the largest integer such that 2^m divides all of the components of y. We call $\theta_k(y)$ the* level *of the point y.*

Given the partition $Z^n = C_k \cup D_k \cup D_{k-1} \cup \cdots \cup D_1$, we have

$$\theta_k(y) = \begin{cases} k, & y \in C_k \\ l-1, & y \in D_l. \end{cases}$$

Definition 2.3 *Let $S \subset Z^n$. Let $\phi(x)$ be an interpolet. Let $\mathcal{I}_k(\phi, S)$ be the space of functions given by formal sums of the form $\sum_{y \in S} a(y)\phi(\frac{x-y}{2^{\theta_k(y)}})$.*

Where ϕ and S are understood, we may simply write $\mathcal{I}_k$.

Definition 2.4 *Let $S \subset Z^n$. Let $\mathcal{F}_k(S)$ be the vector space of R- or C- valued functions on Z^n which are zero at any point not in S (i.e. with support contained in S).*

Where S is understood, we may simply write $\mathcal{F}_k$. Note: $\mathcal{F}_k(S) = \mathcal{F}_k(S \cap C_k) \oplus \mathcal{F}_k(S \cap D_k) \oplus \mathcal{F}_k(S \cap D_{k-1}) \oplus \cdots \oplus \mathcal{F}_k(S \cap D_1)$.

The meaning of the k subscript will be established by the next definition, which will link vectors in $\mathcal{F}_k$ with functions in $\mathcal{I}_k$. It is for this reason that while the $\mathcal{F}$'s are technically identical, they are semantically different. In practice, the $\mathcal{F}$'s are the actual data structures being stored on the computer.

Definition 2.5 *Let $\phi(x)$ be an interpolet. Let $\iota_k^\phi : S \to \mathcal{I}_k(\phi, S)$ be defined by*

$$\iota_k^\phi y = \phi(\frac{x-y}{2^{\theta_k(y)}}).$$

This definition extends linearly to the mapping $\iota_k^\phi : \mathcal{F}_k(S) \to \mathcal{I}_k(\phi, S)$ defined by:

$$\iota_k^\phi v = \sum_{y \in S} v(y)(\iota_k^\phi y)$$

i.e.

$$(\iota_k^\phi v)(x) = \sum_{y \in S \cap C_k} v(y)\phi((x-y)/2^k) + \sum_{y \in S \cap D_k} v(y)\phi((x-y)/2^{k-1}) + \cdots + \sum_{y \in S \cap D_1} v(y)\phi(x-y).$$

The set S can be thought of as the set of points in a refined grid. The ι_k^ϕ identifications allow one to think of S as a set of functions, $\{(\iota_k^\phi y)|y \in S\}$, which form a basis of $\mathcal{I}_k(\phi, S)$. We will sometimes refer to S as a refined grid and sometimes as a basis with this identification understood.

One should think of the $\mathcal{F}_k$ as spaces of coefficients for function expansions in the corresponding $\mathcal{I}_k$ spaces, in the basis S. The ι_k^ϕ simply associate a set of coefficients in $\mathcal{F}_k$ with a function in $\mathcal{I}_k$. When ϕ is understood, we may write just ι_k.

We are now in a position to state the basic theorems of interpolet expansions on uniform grids.

Theorem 2.6 *Let $\phi(x)$ be an interpolet on R^n. Then each mapping $\iota_k : \mathcal{F}_k(S) \to \mathcal{I}_k(\phi, S)$ $(k = 1, 2, \ldots)$ is an isomorphism.*

Proof: Since the map ι_k is surjective, it is only necessary to show that ι_k is injective. By the definition, $\iota_k v = 0$ if and only if there exist $v \in \mathcal{F}_k$ such that

$$0 = \sum_{y \in S \cap C_k} v(y)\phi((x-y)/2^k) + \sum_{y \in S \cap D_k} v(y)\phi((x-y)/2^{k-1}) + \cdots + \sum_{y \in S \cap D_1} v(y)\phi(x-y).$$

Let $z \in S \cap C_k$. By (INT1), we have $\phi((z-y)/2^k) = \delta_{(z-y)/2^k,0} = \delta_{z,y}$, for $y \in S \cap C_k$, and also $\phi((z-y)/2^l) = 0$, for $y \in S \cap D_l$. So, $(\iota_k v)(z) = v(z), z \in S \cap C_k$, therefore $v(z) = 0, z \in S \cap C_k$. This being so, one then has $(\iota_k v)(z) = v(z), z \in S \cap D_k$, so $v(z) = 0, z \in S \cap D_k$. Once again, $(\iota_k v)(z) = v(z), z \in S \cap D_{k-1}$, thus we must have $v(z) = 0, z \in S \cap D_{k-1}$. Continuing in this manner, we deduce that $v(y) = 0, y \in S$, thus $v = 0$.
□

Corollary 2.7

$$\mathcal{I}_k(S) = \mathcal{I}_k(S \cap C_k) \oplus \mathcal{I}_k(S \cap D_k) \oplus \cdots \oplus \mathcal{I}_k(S \cap D_1)$$

Since the sum is direct, the expansion is unique.

This corollary is a consequence of observations in the above proof.

Theorem 2.8 *Let $\phi(x)$ be an interpolet on R^n. Then*

$$\forall k, \mathcal{I}_k(\phi, Z^n) = \mathcal{I}_{k-1}(\phi, Z^n).$$

Consequently,

$$\forall k_1, k_2, \mathcal{I}_{k_1}(\phi, Z^n) = \mathcal{I}_{k_2}(\phi, Z^n).$$

Proof: To prove that $\mathcal{I}_k \subset \mathcal{I}_{k-1}$ we note that $\mathcal{I}_k \subset \mathcal{I}_{k-1} \cup \mathcal{I}_k(C_k)$. Thus we just need to show $\mathcal{I}_k(C_k) \subset \mathcal{I}_{k-1}$.

Translating by $z \in C_k$ and inserting powers of 2 where appropriate, one can rewrite (INT2) for ϕ as

$$\phi((x-z)/2^k) = \phi((x-z)/2^{k-1}) + \sum_{y \in D_k} c_{y/2^{k-1}} \phi((x-z-y)/2^{k-1}).$$

The terms in the right hand side are elements $\mathcal{I}_{k-1}$. Thus $\mathcal{I}_k \subset \mathcal{I}_{k-1}$.

To prove $\mathcal{I}_{k-1} \subset \mathcal{I}_k$ note that any element of $\mathcal{I}_{k-1}$ can be expressed as:

$$f(x) = \sum_{y \in C_{k-1}} a(y)\phi((x-y)/2^{k-1}) + \sum_{y \in D_{k-1}} a(y)\phi((x-y)/2^{k-2}) + \cdots \sum_{y \in D_1} a(y)\phi(x-y)$$

All the terms in this expansion but the first are elements of $\mathcal{I}_k$. Since $C_{k-1} = C_k \cup D_k$ we may split the first sum up as,

$$\sum_{y \in C_{k-1}} a(y)\phi((x-y)/2^{k-1}) = \sum_{y \in C_k} a(y)\phi((x-y)/2^{k-1}) + \sum_{y \in D_k} a(y)\phi((x-y)/2^{k-1})$$

The second term is also an element of $\mathcal{I}_k$.

Rewriting (INT2) one has ($y \in C_k$):

$$\phi((x-y)/2^{k-1}) = \phi((x-y)/2^k) - \sum_{z \in D_k} c_{z/2^{k-1}}\phi((x-z-y)/2^{k-1}).$$

$y \in C_k, z \in D_k$, so $y + z \in D_k$, thus the right hand side is made up of elements of $\mathcal{I}_k$. Thus, $\mathcal{I}_{k-1} \subset \mathcal{I}_k$.

□

2.2 Interpolet Transforms

Corollary 2.9 *Interpolet Decomposition*
The set of isomorphisms ι_k induces a set of isomorphisms

$$J_{k_1,k_2} : \mathcal{F}_{k_1} \to \mathcal{F}_{k_2},$$

$J_{k_1,k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$.

We refer to these isomorphisms as *interpolet transforms*. It is our convention to let $J_k = J_{0,k}$ and $J_{-k} = J_{k,0}$. The reader will note that the J_i are linear transformations on the coefficient spaces, and are thus the primary object of computation.

We now turn to a study of the J's. It is clear from the definition that for $k_1 < k_2$, $J_{k_1,k_2} = J_{k_2-1,k_2} \circ J_{k_2-2,k_2-1} \circ \cdots \circ J_{k_1,k_1+1}$, and similarly for $k_1 > k_2$. Thus, we need only study the $J_{k,k+1}$ and $J_{k+1,k}$ mappings.

Theorem 2.10 *Computation Theorem*
Let $v \in \mathcal{F}_k(Z^n)$.

$$(J_{k,k+1}v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z \in C_{k+1}} c_{(y-z)/2^k} v(z)$.

$$(J_{k+1,k}v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) + \sum_{z \in C_{k+1}} c_{(y+z)/2^k} v(z)$.

Proof: For $v \in \mathcal{F}_k$ we have

$$\iota_k v = \sum_{y\in C_k} v(y)\phi((x-y)/2^k) + \sum_{y\in D_k} v(y)\phi((x-y)/2^{k-1}) + \cdots + \sum_{y\in D_1} v(y)\phi(x-y)$$

expanding the first term,

$$\iota_k v = \sum_{y\in C_{k+1}} v(y)\phi((x-y)/2^k) + \sum_{y\in D_{k+1}} v(y)\phi((x-y)/2^k) + \sum_{y\in D_k} v(y)\phi((x-y)/2^{k-1}) + \cdots$$

using (INT2), $\phi((x-z)/2^k) = \phi((x-z)/2^{k+1}) - \sum_{y\in D_{k+1}} c_{(y-z)/2^k}\phi((x-y)/2^k)$,

$$\iota_k v = \sum_{y\in C_{k+1}} v(y)\phi((x-y)/2^{k+1}) + \sum_{y\in D_{k+1}} v'(y)\phi((x-y)/2^k) + \sum_{y\in D_k} v(y)\phi((x-y)/2^{k-1}) + \cdots$$

$$(\iota_{k+1}^{-1}\iota_k v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z\in C_{k+1}} c_{(y-z)/2^k} v(z)$.

The proof for $J_{k+1,k}$ is similar.

□

Similar to what one might get with wavelets, we see that we can compute the coefficients of interpolet expansions on uniform lattices by a pyramid algorithm. Computationally, this procedure can be carried out by first computing the D_1 coefficients with $J_{0,1}$, then by computing the D_2 coefficients from the C_1 data with $J_{1,2}$, and so on. In this sense, it is no different from standard multiresolution decompositions.

A feature of the interpolet decomposition is that the transformations all have a particular *lower triangular* form. That is, if we write $v \in \mathcal{F}_k$ as a vector with its C_{k+1} components first, its D_{k+1} components second, and the rest of its components third, then the transformation takes the form,

$$J_{k,k+1}v = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ M & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} v_{C_{k+1}} \\ v_{D_{k+1}} \\ v_{D_k} \\ . \\ . \end{pmatrix}.$$

The inverse, $J_{k+1,k}$, is obtained by replacing M with $-M$.

3 Accuracy of Interpolet Approximation

Given a function $f(x)$ on R^n, one can form an interpolet approximation to f by the formula:

$$f(x) \sim \sum_{y\in C_0} f(y)\phi(x-y) = \iota_0 f,$$

where f on the right hand side is thought of as a function restricted to $C_0 = Z^n$ (a more cumbersome but more precise notation is $\iota_0\{f(y)\}|_{y\in Z^n}$). This approximation has the property that $(\iota_0 f)(z) = f(z), z \in Z^n$.

Starting from the expansion $\iota_0 f \in \mathcal{I}_0(\phi, Z^n)$ one can construct equivalent expansions, $\iota_k(J_k f) \in \mathcal{I}_k(\phi, Z^n)$. The coefficients $J_k f$ are referred to as the interpolet transform of the function f.

If $f(x)$ is sufficiently smooth, then we can expect that the coefficients $(J_k f)(y), y \in D_l, l \leq k$, will be small. This statement is captured rigorously by the following lemma and theorem.

Lemma 3.1 *Let ϕ be an interpolet with polynomial order of m then*

$$p(x) \in \mathcal{I}_N(\phi, C_N)$$

for any integer, N, and any polynomial, p, of degree m.

Proof: $p(2^N x)$ is a polynomial of degree m. By (INT3), $p(x)$ can thus be represented by a formal sum in $\mathcal{I}_0(\phi, C_0)$, namely

$$p(2^N x) = \sum_{y \in Z^n} p(2^N y)\phi(x - y).$$

By changing variables, we may rewrite this as

$$\begin{aligned} p(x) &= \sum_{y \in Z^n} p(2^N y)\phi(x/2^N - y) \\ &= \sum_{y \in Z^n} p(2^N y)\phi((x - 2^N y)/2^N) \\ &= \sum_{y \in C_N} p(y)\phi((x - y)/2^N) \\ &= \sum_{y \in C_N} p(y)\phi((x - y)/2^{\theta_N(y)}). \end{aligned}$$

□

Theorem 3.2 *(Residual Theorem)*
Let $f(x)$ be a polynomial function of degree m. Let ϕ be an interpolet with a polynomial order of m. Then

$$(J_k f)(y) = \begin{cases} f(y), & y \in C_k \\ 0, & y \in D_l \end{cases}$$

Proof:
By (INT3), $f(x) \in \mathcal{I}_0(\phi, C_0)$. By the lemma, we also have $f(x) \in \mathcal{I}_k(\phi, C_k)$. Recalling that $J_k f$ gives the unique expansion coefficients of $f(x)$ in the decomposition of $\mathcal{I}_0(\phi, C_0)$ given by $\mathcal{I}_k(\phi, C_k) \oplus \mathcal{I}_k(\phi, D_k) \oplus \cdots \oplus \mathcal{I}_k(\phi, D_1)$, we see that the $\mathcal{I}_k(\phi, D_l)$ coefficients must vanish, while the $\mathcal{I}_k(\phi, C_k)$ coefficients are given by the lemma, namely $f(y)$ for $y \in C_k$.
□

The coefficients $(J_k f)(y), y \in D_l, l \leq k$, are called *residuals*. The Residual Theorem suggests that the magnitude of the residual coefficients $(J_k f)$ at a point $y \in D_l$ are expected to be proportional to the $(m+1)$th derivative of $f(x)$ at y. See Figure 2.

4 Truncated Bases

Typically, one truncates an expansion by eliminating elements of a basis, setting their coefficients to 0. One then said to be working in a truncated basis when one works within the subspace formed by the remain basis elements. In the notation of this paper, this corresponds to taking expansions in $\mathcal{I}_k(\phi, S)$ with coefficients in $\mathcal{F}_k(S)$.

One may also view a truncated basis as the set of expansions one gets when the coefficients of some of the basis elements have been set to "don't care" values. Mathematically, this is accomplished by quotienting out the "don't care" elements.

Definition 4.1 *Let*

$$\mathcal{I}_k^S = \mathcal{I}_k(\phi, Z^n)/\mathcal{I}_k(\phi, Z^n - S)$$

and

$$\mathcal{F}_k^S = \mathcal{F}_k(Z^n)/\mathcal{F}_k(Z^n - S).$$

When the identification of $F_k(S)$ with F_k^S can be made will be dealt with later in this paper. For now, one may view these definitions as a trick to make the proofs less complicated and for understanding exactly why and in what sense the algorithms are correct. Once again, we think of $\mathcal{F}_k^S$ as a grid on which the elements outside of S have "don't care" values, and $\mathcal{F}_k(S)$ as a grid on which the elements outside of S vanish. The ι_k continue to be isomorphisms (since $\iota_k(\mathcal{F}_k(Z^n - S)) = \mathcal{I}_k(\phi, Z^n - S)$).

However, it is not necessarily true that $\mathcal{I}_{k_1}^S = \mathcal{I}_{k_2}^S$. When this condition fails, then it is no longer possible to define $J_{k_1,k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$.

To be sure, one could still define some sort of J_{k_1,k_2} by setting the elements in $Z^n - S$ to zero, then applying the full grid version of J_{k_1,k_2}, and then considering only the elements in S of the answer. This definition by itself has some drawbacks. Mathematically speaking, this is the same as $J_{k_1,k_2} = p \circ \iota_{k_2}^{-1} \circ \iota_{k_1} \circ r$, where $r : \mathcal{F}_{k_1}^S \to \mathcal{F}_{k_1}$, is some lift to the full grid, and $p : \mathcal{I}_{k_2} \to \mathcal{I}_{k_2}^S$ is the standard projection operator onto the quotient. Generally, if one follows this approach one will no longer have $J_{k_1,k_2} = J_{k_2-1,k_2} \circ J_{k_2-2,k_2-1} \circ \cdots \circ J_{k_1,k_1+1}$ because $\mathcal{I}_i^S$ are not all equal. In terms of diagrams, where we once had

$$\mathcal{F}_{k_1} \overset{\iota_{k_1}}{\to} \mathcal{I}_{k_1} = \mathcal{I}_{k_2} \overset{\iota_{k_2}^{-1}}{\to} \mathcal{F}_{k_2},$$

we now have

$$\mathcal{F}_{k_1}^S \overset{\iota_{k_1}}{\to} \mathcal{I}_{k_1}^S \overset{?}{=} \mathcal{I}_{k_2}^S \overset{\iota_{k_2}^{-1}}{\to} \mathcal{F}_{k_2}^S.$$

What one needs is a condition on S such that $\mathcal{I}_{k_1}(Z^n - S) = \mathcal{I}_{k_2}(Z^n - S)$. If this were true, then the definition of the operator as $J_{k_1,k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$ would actually be independent of the values of the elements of $Z^n - S$. In that case the quotient spaces are identical.

Definition 4.2 *We say that the set S is a* good basis *when it satisfies the condition* $\forall k_1, k_2, \mathcal{I}_{k_1}(Z^n - S) = \mathcal{I}_{k_2}(Z^n - S)$*, and thus* $\mathcal{I}_{k_1}^S = \mathcal{I}_{k_2}^S$.

To get a handle on this definition, one sees that this is achieved when $\mathcal{I}_k(Z^n - S) = \mathcal{I}_{k+1}(Z^n - S)$. For this to be so, whenever $y \in Z^n - S$, every z such that $\phi(x/2^{\theta_N(z)} - z)$ is in the two-scale expansion for $\phi(x/2^{\theta_N(y)} - y)$, must also be a member of $Z^n - S$.

This can be captured in the following table (in which we let $\theta_k(y) = \theta_k(z) + 1$).

in expansion?	$z \in S$	$z \in Z^n - S$
$y \in S$	ok	ok
$y \in Z^n - S$	not ok	ok

The good basis condition for fast synthesis and reconstruction has also been discovered by Cohen and Danchin (see *S-trees* in a coming work[10]) which appeared after the original submission of our manuscript.

For some of the algorithms presented, we may employ additional conditions based on the supports of the functions themselves (not just the support of their expansions).

Definition 4.3 *We say that functions f and g* touch *whenever $supp\{f\} \cap supp\{g\}$ has nonzero measure.*

*For $p > 0$ we say that S has the p-*level touching *property when it satisfies the condition, that for $y \in Z^n - S$, $z \in Z^n, \theta_k(z) \leq \theta_k(y) - p$, and $\iota_k y$ touches $\iota_k z$ implies $z \in Z^n - S$.*

A less formal way of phrasing this definition for the case of 1-level touching is that if a level l point,y, is a member of $Z^n - S$ then any point, z, at level $l-1$ or lower for which $\iota_k y$ touches $\iota_k z$ must also be in $Z^n - S$. For 2-level touching, one only considers any points at level $l-2$ or lower, and so on for p-level touching.

The allowed touching possibilities can be summarized the following table (in which we let $\theta_k(y) \geq \theta_k(z) + p$).

touch?	$z \in S$	$z \in Z^n - S$
$y \in S$	ok	ok
$y \in Z^n - S$	not ok	ok

One example of a 1-level touching good basis for one dimensional 3rd order interpolets (the ones being used as an example) is the set of interpolets centered at the points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-7 \cdot 2^l, -5 \cdot 2^l, \ldots, 5 \cdot 2^l, 7 \cdot 2^l\} \right)$$

The support of the interpolet on level 0 at y is $[-3 + y, 3 + y]$, the union of all the supports of the interpolets in S on level 0 is $[-10, 10]$. The support of an interpolet on level 1 at y is $[-6 + y, 6 + y]$, thus the only interpolets on level 1 which touch the interpolets on level 0 are precisely those ones at points $-14, \ldots, 14$, which are precisely the ones included in S. No interpolet not included on level 1 touches an interpolet included on level 0 so the definition is satisfied. The argument proceeds similarly on higher levels. In three dimensions, this example corresponds to nested concentric cubes of size $15^3 \cdots 2^l$ at each level $l < n$.

An example of a 2-level touching good basis for 3rd order is the set of interpolets centered at the points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-5 \cdot 2^l, -3 \cdot 2^l, \ldots, 3 \cdot 2^l, 5 \cdot 2^l\} \right).$$

Note (for $n \geq 2$) that this set is not 1-level touching since the level 1 interpolet centered at $y = 12$ is not included, while an interpolet it touches, namely the level 0 interpolet centered at $y = 5$, is included.

Also, the set of points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-3 \cdot 2^l, -1 \cdot 2^l, 1 \cdot 2^l, 3 \cdot 2^l\} \right).$$

forms a good basis, but is not 1-level touching or 2-level touching (it is 3-level touching, though).

The above examples are meant to suggest that the *good basis* and the *p-level touching* definitions can be thought of, informally, as conditions telling one how quickly one can change from one resolution to another. Essentially, any nested set of refined regions can satisfy these conditions so long as the margins around the set of points at a given resolution are wide enough.

¿From a computational point of view, what these conditions do is ensure that data-paths which carry coefficient information between different resolutions are not broken by zeroed coefficients at intermediate levels.

It is clear from the preceding discussion, in a good basis, one has $J_{k_1,k_2} = J_{k_2-1,k_2} \circ J_{k_2-2,k_2-1} \circ \cdots \circ J_{k_1,k_1+1}$. We may now generalize the computation theorem to a truncated basis.

Theorem 4.4 *Good Basis Computation Theorem*
Let S be a good basis. $v \in \mathcal{F}_k(S)$, $y \in S$
and $\tilde{v}$ is any member of the equivalence class of v

$$(J_{k,k+1}\tilde{v})(y) = \begin{cases} v(y), & y \in S - D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z \in S \cap C_{k+1}} c_{(y-z)/2^k} v(z)$.

$$(J_{k+1,k}\tilde{v})(y) = \begin{cases} v(y), & y \in S - D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) + \sum_{z \in S \cap C_{k+1}} c_{(y+z)/2^k} v(z)$.

Proof:

In a good basis, the computations of all J_{k_1,k_2} are independent of the representative. Thus, this algorithm computed on $\tilde{v}$ gives a member of the same class as would be computed on v.

$\square$

Thus, the pyramid algorithm of the uniform basis, Z^n, has a counterpart in a good basis S, allowing the computation of the expansion coefficients in $\mathcal{F}_k(S)$ from the values of the function in $\mathcal{F}_0(S)$ (and also has the lower triangular structure).

In a good basis, one thus has the ability to perfectly reconstruct the multiscale coefficients of a function for the basis functions associated with the points of the refined grid S by simply applying the pyramid algorithm on zero-lifts at each stage of the algorithm. The above theorem establishes this as true, even though we do not necessarily expect the data zeroed during the lift to be small. (The function may have significant sample values throughout the domain of the representation). Also, with exact recovery of sample values, it is easy to perform local nonlinear point-wise operations of the form $f(x) \to G(f(x))$ (e.g. $e^{f(x)}$), or point-wise multiplication (i.e. $f(x), g(x) \to f(x)g(x)$), despite the truncation.

The reader may note that this result is "analysis-free" in that we have sparsified the computation, not by proving that the coefficients vanish outside of the truncation for some class of functions, but by showing the the coefficients we wish to compute have no dependency on the coefficients we omitted. Computationally, this means the data-structure in the computer requires no intermediate augmentations (contrast with [18]).

The advantages conferred by the additional the p-level properties are seen in the context of operator evaluation, and will be the subject of the next two sections.

5 Multilevel Algorithms for ∇^2 and Other Operators

Given $v, w \in \mathcal{F}_k(Z^n)$, we may compute the stiffness matrix of a model operator, ∇^2,

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle = \int (\iota_k v)(x)\nabla^2(\iota_k w)(x)d^n x$$

by changing the expansions,

$$\left\langle \iota_0 J_{-k} v|\nabla^2|\iota_0 J_{-k} w\right\rangle = \sum_{y,z\in Z^n} (J_{-k}v)(y)(J_{-k}w)(z)\int \phi(x-y)\nabla^2\phi(x-z)d^n x.$$

This reduces the computation to the computation of the matrix elements $\langle\phi(x-(y-z))|\nabla^2|\phi(x)\rangle$ which can be done by solving the associated eigen-problem obtained by applying (INT2).

In particular, let $L^0_y = \langle\phi(x-y)|\nabla^2|\phi(x)\rangle$, then

$$L^0_y = \sum_{z_1,z_2\in Z^n} c_{z_1}c_{z_2}\left\langle\phi(2x-2y-z_1+z_2)|\nabla^2|\phi(2x)\right\rangle$$

$$L^0_y = \sum_{z_1,z_2\in Z^n} 2^{2-n}c_{z_1}c_{z_2}L^0_{2y+z_1-z_2}$$

which we solve by standard methods (found in [25], for example). In subsequent sections of this article, we will also define $L^+_y = \langle\phi(x-y)|\nabla^2|\phi(x/2)\rangle$, $L^{++}_y = \langle\phi(x-y)|\nabla^2|\phi(x/4)\rangle$, $L^-_y = \langle\phi((x-y)/2)|\nabla^2|\phi(x)\rangle$, and $L^{--}_y = \langle\phi((x-y)/4)|\nabla^2|\phi(x)\rangle$, which can be computed from L^0 by employing (INT2). Although it is true by Hermiticity that $L^-_y = L^+_y$ and $L^{--}_y = L^{++}_y$, we will make no use of this fact.

One can write a matrix expression $\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t J_{-k}^t L J_{-k} w$, where the L is the Toeplitz matrix with coefficients L_y^0. In practice, one typically formulates the preceding as a computation of $u = J_{-k}^t D J_{-k} w$ for some $w \in \mathcal{F}_k(Z^n)$. Then $\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t u$. u may be thought of as an element of $\mathcal{F}_k(Z^n)^*$, the dual space of $\mathcal{F}_k(Z^n)$. The task is then to compute the coefficients $u(y), y \in Z^n = C_k \oplus D_k \oplus \cdots \oplus D_1$. Any algorithm for computing $x^t A y$ can be adapted to an algorithm for Ay, and for purposes of making proofs, it is somewhat easier to keep thinking of the computation as $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$, which is the point of view we shall take.

However, computing $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle = \tilde{v}^t J_{-k}^t L J_{-k} \tilde{w}$, by just applying the transforms, and the Toeplitz matrix is problematic, since this process makes it necessary to either represent v and w on a uniform grid, or to compute a matrix element between each pair of functions in the truncated expansion which touch. In the first case, one ends up with an $O(N)$ computation for (typically) a very large N. In the second case, one chooses between extremes which are $O(N)$ and quite complicated or simple and $O(N^2)$.

The following sections outline the design of multilevel algorithms for ∇^2 for both 1-level and 2-level touching bases. Both algorithms are derived according to the following format:

break up the expansion of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ into a decomposition over elements at the same level and adjacent levels.

rewrite the expansions in terms of the matrix elements between elements of those levels and the transforms of higher/lower level elements.

implement the algorithm by computing those terms separately.

establish correctness in a p-level truncated basis.

Although only the 1-level and 2-level algorithms have been explored in any detail, this same process will generally work to produce $O(N)$ p-level algorithms for any p.

5.1 ∇^2 in 1-level decomposition

The 1-level decomposition of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ is

$$\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle = \left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^+ + \left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^0 + \left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^- ,$$

where

$$\begin{aligned}
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^0 &= \sum_{\theta_k(y) = \theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^+ &= \sum_{\theta_k(y) < \theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^- &= \sum_{\theta_k(y) > \theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z).
\end{aligned}$$

That is, we express the product as contributions from levels to the same level, higher levels, and lower levels. We will now investigate each of these terms individually. In the language of matrices, these respectively correspond to diagonal, above diagonal, and below diagonal blocks of the ∇^2 matrix.

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle = v^t \begin{pmatrix} \langle|\rangle^0 & \langle|\rangle^+ & \langle|\rangle^+ & \langle|\rangle^+ \\ \langle|\rangle^- & \langle|\rangle^0 & \langle|\rangle^+ & \langle|\rangle^+ \\ \langle|\rangle^- & \langle|\rangle^- & \langle|\rangle^0 & \langle|\rangle^+ \\ \langle|\rangle^- & \langle|\rangle^- & \langle|\rangle^- & \langle|\rangle^0 \end{pmatrix} w$$

5.1.1 Diagonal blocks: $\langle|\rangle^0$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=\theta_k(y)=l} \left\langle \phi((x-y)/2^l)|\nabla^2|\phi((x-z)/2^l)\right\rangle v(y)w(z)$$

contributes to $\langle|\rangle^0$.

However, $\left\langle \phi((x-y)/2^l)|\nabla^2|\phi((x-z)/2^l)\right\rangle = 2^{l(n-2)} \langle \phi(x-y)|\nabla^2|\phi(x-z)\rangle = 2^{l(n-2)} L^0_{z-y}$. Thus we have

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^0 = \sum_l 2^{l(n-2)} \sum L^0_{z-y} v(y)w(z).$$

5.1.2 Super-diagonal blocks: $\langle|\rangle^+$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=l'>l=\theta_k(y)} \left\langle \phi((x-y)/2^l)|\nabla^2|\phi((x-z)/2^{l'})\right\rangle v(y)w(z)$$

contributes to $\langle|\rangle^+$.

Applying the inverse transform $(J_{k,l+1})$ to the w coefficients in the sum allows us to write this term as

$$\sum_{\theta_{l+1}(z)=l+1>l=\theta_k(y)} \left\langle \phi((x-y)/2^l)|\nabla^2|\phi((x-z)/2^{l+1})\right\rangle v(y)(J_{k,l+1}w)(z).$$

Thus we have

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^+ = \sum_l 2^{l(n-2)} \sum L^+_{z-y} v(y)(J_{k,l+1}w)(z).$$

5.1.3 Sub-diagonal blocks: $\langle|\rangle^-$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=l<l'=\theta_k(y)} \left\langle \phi((x-y)/2^{l'})|\nabla^2|\phi((x-z)/2^l)\right\rangle v(y)w(z)$$

contributes to $\langle|\rangle^-$.

This time applying the inverse transform $(J_{k,l+1})$ to the v coefficients in the sum allows us to write this term as

$$\sum_{\theta_k(z)=l<l+1=\theta_{l+1}(y)} \left\langle \phi((x-y)/2^{l+1})|\nabla^2|\phi((x-z)/2^l)\right\rangle (J_{k,l+1}v)(y)w(z).$$

Thus we have

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^- = \sum_l 2^{l(n-2)} \sum L^-_{z-y}(J_{k,l+1}v)(y)w(z).$$

5.1.4 Implementations

The above observations demonstrate the correctness of the following algorithm:
input: $v, w \in \mathcal{F}_k(Z^n)$
output: $ans = \langle \iota_k v|\nabla^2|\iota_k w\rangle \in R$

Let $wtmp = w$, let $vtmp = v$, let $ans = 0$
for $l = 0$ to k
    $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L^0_{z-y} v(y)w(z)$
end for
for $l = k-1$ down-to 0
    $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l,\theta_{l+1}(z)=l+1} L^+_{z-y} v(y)wtmp(z)$
    $wtmp \leftarrow J_{l+1,l}(wtmp)$
end for
for $l = k-1$ down-to 0
    $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_{l+1}(y)=l+1,\theta_k(z)=l} L^-_{z-y} vtmp(y)w(z)$
    $vtmp \leftarrow J_{l+1,l}(vtmp)$
end for

Note, we have made use of the fact that $J_{k,l} = J_{l+1,l}\cdots J_{k,k-1}$ so that at the beginning of each iteration in the second (last) loop $wtmp = J_{k,l+1}w$ ($vtmp = J_{k,l+1}v$).

We observe that this algorithm is $O(N)$ in time and space.

We may adapt this to an algorithm to compute $u \in \mathcal{F}_k(Z^n)^*$ such that $u(v) = \langle \iota_k v|\nabla^2|\iota_k w\rangle$.
input: $w \in \mathcal{F}_k(Z^n)$
output: u such that $u(v) = \langle \iota_k v|\nabla^2|\iota_k w\rangle$

Let $wtmp = w$, let $u = 0 \in \mathcal{F}_k(Z^n)^*$, let $utmp = 0 \in \mathcal{F}_0(Z^n)^*$
for $l = 0$ to k
    $u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=level_k(z)=l} L^0_{z-y} w(z)$
end for
for $l = k-1$ down-to 0
    $u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l,level_{l+1}(z)=l+1} L^+_{z-y} wtmp(z)$
    $wtmp \leftarrow J_{l+1,l} wtmp$
end for
for $l = 0$ to $k-1$
    $utmp \leftarrow J^t_{l+1,l} utmp$
    $utmp(y) \leftarrow utmp(y) + 2^{l(n-2)} \sum_{\theta_{l+1}(y)=l+1,\theta_k(z)=l} L^-_{z-y} w(z)$

end for

$u \leftarrow u + utmp$

The third loop is the result of transposing the linear operator $J_{l+1,l}$ in the last loop in the previous algorithm. We have also used the fact that $J^t_{k,l} = J^t_{k,k-1} \cdots J^t_{l+1,l}$ to ensure that at the beginning of each iteration in the third loop, $utmp \in \mathcal{F}_{l+1}(Z^n)^*$.

It is easy to check that this is an $O(N)$ algorithm in both time and space. It is very important for atomic structures computation that this algorithm scales linearly with the number of atoms. Without such a scaling, one can only compute electronic configurations for small molecules.

The reader may note a similarity between this algorithm and other matrix-vector multiplies used to apply operators in a uniform wavelet basis. In fact, the 1-level algorithm presented above is identical to the nonstandard multiply found in [4] and developed for orthonormal wavelet bases. The nonstandard multiply was introduced by Beylkin, Coifman, and Rokhlin to sparsify integral operators whose kernels were smooth or vanishing off the diagonal, while keeping a uniform basis.

However, in contrast to that program of sparsification, interpolets allow one to sparsify the basis, and, with out introducing additional grid points, still be able to apply the nonstandard multiply routines, with any local operator. With interpolets, we remove any elements from the expansion that we believe will be insignificant, still having a good approximation to our function at the points we retain. Beylkin *et al.* [4] express the matrix elements of the operator itself in a nonstandard orthonormal basis and then remove those matrix elements which are determined to be very small to produce a sparse matrix.

The use of interpolating scaling functions has achieved some degree of simplicity and convenience in carrying out fast point-wise operations. Although there is no associated difficulty in electronic structure calculations[1], for other applications, the loss of orthogonality might be too great an expense. In those cases, one might consider employing compactly supported approximations to orthogonal interpolating functions found in [5]. It appears that with some additional complexity one might be able to extend the present algorithms to other wavelet bases. The additional complexity of other schemes and the need for fast point-wise operations in our applications are the chief reasons we do not consider doing this in the present work. Finally, there is a large body of work the reader may wish to consult ([6], [17], [11], [21], and [22]) for adaptive refinement techniques when, in contrast to the case of electronic structure calculations, the behavior of the needed refinement is not known *a priori*.

5.1.5 Correctness in a 1-level touching good basis

The above decomposition of the product and the associated algorithm is what we seek to extend to a good truncated basis. In practice, one takes the zero-lift representatives of v and $w \in \mathcal{F}^S_k$ and computes $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle$. By the computation theorem of good truncated bases, the value of $(J_{k_1,k_2}\tilde{v})(y), y \in \mathcal{F}^S_k$ is independent of the representative (likewise for w), however, we must also address the issue that $(J_{k_1,k_2}\tilde{v})(y) \neq 0, y \notin S$ (i.e. $J_{k_1,k_2}\tilde{v} \neq \widetilde{J_{k_1,k_2}v}$), and thus $y \notin S$ may have a contribution to the decomposition above, requiring us to augment S in order to get the right answer.

Theorem 5.1 *If one replaces Z^n with S everywhere in the Multilevel algorithm for $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$,* *then the algorithm computes*

$$\left\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \right\rangle .$$

Proof: As mentioned in the remarks, the multilevel algorithm requires $J_{k,l} = J_{l+1,l} \cdots J_{k,k-1}$, which is true in a good bases.

The $\langle | \rangle^0$ computation proceeds identically for either Z^n or S, so the first loop will contribute correctly the term $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle^0$.

To check the $\langle | \rangle^+$ contribution, one observes that the necessary term is

$$\left\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \right\rangle^- = \sum_l 2^{l(n-2)} \sum_{y \in Z^n, z \in Z^n : \theta_{l+1}(y) = l+1, \theta_k(z) = l} L^-_{z-y} (J_{k,l+1} \tilde{v})(y) \tilde{w}(z).$$

Suppose that $\exists y \notin S$ and $z \in S$ such that $L^+_{z-y} \neq 0$. This implies that $\mathrm{supp}\{\iota_{l+1} y\} \cap \mathrm{supp}\{\iota_k z\}$ has nonzero measure. Since $\mathrm{supp}\{\iota_{l+1} y\} \subset \mathrm{supp}\{\iota_k y\}$ we conclude that $\mathrm{supp}\{\iota_k y\} \cap \mathrm{supp}\{\iota_k z\}$ has nonzero measure. This cannot be so if S is 1-level touching, and since $\tilde{w}(z) = 0, z \notin S$ we may restrict the sums over y and z in the contribution,

$$2^{l(n-2)} \sum_{y \in S, z \in S : \theta_{l+1}(y) = l+1, \theta_k(z) = l} L^-_{z-y} (J_{k,l+1} \tilde{v})(y) \tilde{w}(z).$$

The proof for $\langle | \rangle^-$ is identical with v's and w's reversed.

□

Immediately we have the following:

Corollary 5.2 *If one replaces Z^n with S everywhere in the Multilevel algorithm for u such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$, then the algorithm computes*

$$u \in (\mathcal{F}^S_k)^*, u(v) = \left\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \right\rangle .$$

The computation of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ serves as a template for another common computation one may wish to perform, namely $\langle \iota_k v | \iota_k w \rangle = \int (\iota_k v)(x)(\iota_k w)(x) d^n x$, i.e. the $L^2(R^n)$ inner product of $\iota_k v$ and $\iota_k w$.

5.2 Computing Other Operators

To compute $\langle \iota_k \tilde{v} | \iota_k \tilde{w} \rangle$, one simply replaces L^0, L^+, and L^- with $G^0_y = \langle \phi(x-y) | \phi(x) \rangle$, $G^+_y = \langle \phi(x-y) | \phi(x/2) \rangle$, and $G^-_y = \langle \phi((x-y)/2) | \phi(x) \rangle$, and then replaces the factors of $2^{l(n-2)}$ with 2^{ln}. After that, the algorithms and theorems for ∇^2 carry over directly.

The above procedure can be used for creating a multilevel algorithm for any operator, $\mathcal{O}$, which is

local : $\mathrm{supp}\{\mathcal{O} f\} \subset \mathrm{supp}\{f\}$

translation invariant : $\mathcal{O} f(x+a) = (\mathcal{O} f)(x+a)$

homogeneous :$\mathcal{O}f(sx) = s^\alpha(\mathcal{O}f)(sx)$
by forming the appropriate coefficients, $\mathcal{O}_y^{\{0,+,-\}}$, and inserting appropriate factors of $2^{l(\alpha+d)}$.

However, locality is the only property which is really required for $O(N)$ multilevel algorithms so long as one can compute $\mathcal{O}_{l,m,m'}^{\{0,+,-\}}$ efficiently.

As an example of a local, homogeneous, but not translationally invariant operator, we shall discuss the coefficients for the multilevel algorithm for the $\hat{x_1}$ operator in two dimensions.

We first consider the coefficients $\hat{x_1}_{l,m,m'}^{\{0,+,-\}}$ given by

$$\hat{x_1}_{l,m,m'}^{\{0,+,-\}} = \int \phi((x_1 - m_1)/2^{l(+1)})\phi((x_2 - m_2)/2^{l(+1)})x_1\phi((x_1 - m_1')/2^{l(+1)})\phi((x_2 - m_2')/2^{l(+1)})dx_1dx_2,$$

i.e.

$$\begin{aligned}
\hat{x_1}_{l,m,m'}^{0} &= \int \phi((x_1 - m_1)/2^{l})\phi((x_2 - m_2)/2^{l})x_1\phi((x_1 - m_1')/2^{l})\phi((x_2 - m_2')/2^{l})dx_1dx_2 \\
\hat{x_1}_{l,m,m'}^{+} &= \int \phi((x_1 - m_1)/2^{l})\phi((x_2 - m_2)/2^{l})x_1\phi((x_1 - m_1')/2^{l+1})\phi((x_2 - m_2')/2^{l+1})dx_1dx_2 \\
\hat{x_1}_{l,m,m'}^{-} &= \int \phi((x_1 - m_1)/2^{l+1})\phi((x_2 - m_2)/2^{l+1})x_1\phi((x_1 - m_1')/2^{l})\phi((x_2 - m_2')/2^{l})dx_1dx_2.
\end{aligned}$$

Separating the x_1 and x_2 integrations, we see from this that we may write $\hat{x_1}_{l,m,n}^{\{0,+,-\}} = 2^{l(d-1)}G_{(m_1-n_1)/2^l}^{\{0,+,-\}}X_{l,m_2,n_2}^{\{0,+,-\}}$ where

$$X_{l,m,n}^{\{0,+,-\}} = \int \phi((x-m)/2^{l(+1)})x\phi((x-n)/2^{l(+1)})dx$$

and G is defined above. The problem of computing the $\hat{x_1}$ coefficients has been reduced to computing the X coefficients and then doing a multiply with the already known G coefficients.

In addition, one has

$$\int \phi((x-m)/2^{l(+1)})x\phi((x-n)/2^{l(+1)})dx = n\int \phi((x+n-m)/2^{l(+1)})\phi(x/2^{l(+1)})dx + \int \phi((x+n-m)/2^{l(+1)})x\phi(x/2^{l(+1)})dx,$$

and thus

$$\int \phi((x-m)/2^{l(+1)})x\phi((x-n)/2^{l(+1)})dx = n2^lG_{(m-n)/2^l}^{\{0,+,-\}} + 2^{2l}S_{(m-n)/2^l}^{\{0,+,-\}}$$

where

$$\begin{aligned}
S_y^0 &= \int \phi(x-y)x\phi(x)dx \\
S_y^+ &= \int \phi(x-y)x\phi(x/2)dx \\
S_y^- &= \int \phi((x-y)/2)x\phi(x)dx.
\end{aligned}$$

Thus we see that for the $\hat{x_1}$ operator,

$$\hat{x_1}_{l,m,n}^{\{0,+,-\}} = 2^lG_{(m_2-n_2)/2^l}^{\{0,+,-\}}(n2^lG_{(m-n)/2^l}^{\{0,+,-\}} + 2^{2l}S_{m-n}^{\{0,+,-\}}),$$

giving an efficient means to compute the multilevel coefficient for this operator.

5.3 ∇^2 in 2-level decomposition

The previous algorithm for 1-level touching good bases can be expanded to 2-level touching good bases. One may wish to do this because one finds that the 1-level touching property is too stringent and requires one to augment one's basis set far too much to be practical computationally.

Much of the reasoning for the 2-level case can be found in the details of the 1-level case, so the exposition here will be more compact. The resulting algorithm will be correct for 2-level touching good bases.

The 2-level decomposition of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ is

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle = \langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^-$$
$$+ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{++} + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{--}$$

where

$$\begin{aligned}
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^0 &= \sum_{\theta_k(y)=\theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^+ &= \sum_{\theta_k(y)+1=\theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^- &= \sum_{\theta_k(y)-1=\theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^{++} &= \sum_{\theta_k(y)+1<\theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z) \\
\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^{--} &= \sum_{\theta_k(y)-1>\theta_k(z)} \left\langle \iota_k y | \nabla^2 | \iota_k z \right\rangle v(y) w(z)
\end{aligned}$$

Which corresponds to the matrix decomposition:

$$\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle = v^t \begin{pmatrix} \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^{++} & \langle | \rangle^{++} \\ \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^{++} \\ \langle | \rangle^{--} & \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ \\ \langle | \rangle^{--} & \langle | \rangle^{--} & \langle | \rangle^- & \langle | \rangle^0 \end{pmatrix} w.$$

The key idea is to evaluate the diagonal and first off diagonal blocks of the ∇^2 matrix and then to compute the other blocks above and below the tridiagonal through the transforms.

5.3.1 $\langle | \rangle^0$, $\langle | \rangle^+$, $\langle | \rangle^-$, $\langle | \rangle^{++}$, and $\langle | \rangle^{--}$

The definitions for the contributions in the decomposition proceed just as they did for the 1-level case.

$$\left\langle \iota_k v | \nabla^2 | \iota_k w \right\rangle^0 = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=\theta_k(y)=l} L^0_{z-y} v(y) w(z).$$

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^+ = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)-1=\theta_k(y)=l} L^+_{z-y} v(y)w(z).$$

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^- = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=\theta_k(y)-1=l} L^-_{z-y} v(y)w(z).$$

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^{++} = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)-1>l=\theta_k(y)} L^{++}_{z-y} v(y)(J_{k,l+2}w)(z).$$

$$\left\langle \iota_k v|\nabla^2|\iota_k w\right\rangle^{--} = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=l<\theta_k(y)-1} L^{--}_{z-y} (J_{k,l+2}v)(y)w(z).$$

5.3.2 Implementations

input: $v, w \in \mathcal{F}_k(Z^n)$
output: $ans = \langle \iota_k v|\nabla^2|\iota_k w\rangle \in R$
    Let $wtmp = w$, let $vtmp = v$, let $ans = 0$
    for $l = 0$ to k
        $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L^0_{z-y} v(y)w(z)$
    end for
    for $l = 0$ to $k-1$
        $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l,\theta_k(z)=l+1} L^+_{z-y} v(y)w(z)$
    end for
    for $l = 0$ to $k-1$
        $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l+1,\theta_k(z)=l} L^-_{z-y} v(y)w(z)$
    end for
    for $l = k-2$ down-to 0
        $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l,\theta_{l+2}(z)=l+2} L^{++}_{z-y} v(y)wtmp(z)$
        $wtmp \leftarrow J_{l+1,l} wtmp$
    end for
    for $l = k-2$ down-to 0
        $ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_{l+2}(y)=l+2,\theta_k(z)=l} L^{--}_{z-y} vtmp(y)w(z)$
        $vtmp \leftarrow J_{l+1,l} vtmp$
    end for

We adapt this algorithm to compute $u \in \mathcal{F}_k(Z^n)^*$ such that $u(v) = \langle \iota_k v|\nabla^2|\iota_k w\rangle$.
input: $w \in \mathcal{F}_k(Z^n)$
output: u such that $u(v) = \langle \iota_k v|\nabla^2|\iota_k w\rangle$
    Let $wtmp = w$, let $u = 0 \in \mathcal{F}_k(Z^n)^*$, let $utmp = 0 \in \mathcal{F}_0(Z^n)^*$
    for $l = 0$ to k
        $u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L^0_{z-y} w(z)$
    end for
    for $l = 0$ to $k-1$
        $u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l,\theta_k(z)=l+1} L^+_{z-y} w(z)$

end for
for $l = 0$ to $k-1$
$\quad u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l+1, \theta_k(z)=l} L^{-}_{z-y} w(z)$
end for
for $l = k-2$ down-to 0
$\quad u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_{l+2}(z)=l+2} L^{++}_{z-y} wtmp(z)$
$\quad wtmp \leftarrow J_{l+1,l} wtmp$
end for
for $l = 0$ to $k-2$
$\quad utmp \leftarrow J^t_{l+1,l} utmp$
$\quad utmp(y) \leftarrow utmp(y) + 2^{l(n-2)} \sum_{\theta_{l+2}(y)=l+2, \theta_k(z)=l} L^{--}_{z-y} w(z)$
end for
$u \leftarrow u + utmp$

6 Efficient Implementation

We have produced a very successful 3D implementation of all of the above algorithms for the interpolet used as this paper's example ($m = 3$). Implementation details are given in this section. The ideas used to make this implementation efficient for all of the above algorithms.

The purpose of this section is to give additional information to readers who wish to implement these algorithms themselves.

6.1 Data Structures

The interpolet data and function samples are kept in a sequence of blocks at various levels. Each block at level k contains the points of a rectangular subset of C_{k-1}. Since $D_k = C_{k-1} - C_k$, we use the collection of blocks at level $k < p$ (p being the top level) to represent a rectangular subset O_k, ignoring the C_k points of each of these blocks. In our implementation, these extra C_k points hold the value 0 in between operations and take on useful intermediate values during operations. Since we are working in 3 dimensions, this multiplies the storage required by a factor of about $\frac{8}{7}$, which we found an acceptable cost for its advantages.

The coefficients for the transforms and operators are kept in various 3D arrays. Although it is possible to build the coefficients upon demand from a set of 1D arrays of coefficients, we have found that the arithmetic cost of doing this is much greater than the cost of storing them (about 10 flops are required for each ∇^2 coefficient, while the 3D arrays are still small enough to be stored in cache). We have (in Fortran notation) the filters cs(0:3,0:3,0:3), SAMELEVEL(0:5,0:5,0:5), ONELEVEL(0:8,0:8,0:8), and (for 2-level algorithms) TWOLEVEL(0:14,0:14,0:14) (note: we have made use of the fact that our operators are symmetric to cut the size of these arrays by a factor of $\frac{1}{8}$ and use ONELEVEL and TWOLEVEL for both upward and downward inter-level communication).

6.2 Implementation Overview

The blocks described in the previous section are used as the fundamental objects for manipulation. The computation proceeds by employing block-to-block subroutines for the various operations, having every block at a level send data to every block at the same level, one level up or down, or (for 2-level algorithms) two levels up or down.

The number of blocks at each level is not very large, and if a subroutine determines that the intersection of two blocks is empty (which it does by examining the bounding rectangles), then it returns immediately. Thus, while this algorithm is to be $O(B^2)$ where B is the number of blocks, it remains $O(N)$ where N is the number of actual points, because B is much smaller than N.

6.3 block-to-block Subroutines

The block-to-block subroutines are all designed to take two blocks (source and destination) and a set of filter coefficients and place the result of convolving the filter with the source block in the overlapping points of the destination block. There is a block-to-block subroutine for the interpolet transform, its transpose, its inverse, and its inverse transpose, as well as operator application routines for the same-level operator, up-one-level operator, down-one-level operator, up-two-level operator, and the down-two-level operator.

All of these routines precompute the bounding box for the points in the destination block which are in the range of influence of the source block and, for each point in this sub-block, the bounding box for the points in the source block in the domain of influence of the destination point. The result of this precomputation is that the only data values of the source (destination) which are accessed are the ones which are read (modified). This decreases the number of data accesses in our test problems by a factor of 7.

Additionally, blocking the computation generally increases the locality of access for the data. More data requests hit the cache than would occur in a more arbitrarily arranged construction.

7 Results and Conclusions

With interpolets, it is possible to carry out $O(N)$ computations in truncated bases, where N is the number of elements retained in the truncation, without having to augment the grid of points associated with the functions maintained in the basis. Along with allowing one to compute common linear physical operations, interpolet algorithms also allow one to transfer between function values and multiscale expansion coefficients on grids of variable resolution recovering the same results as one would obtain working with data on a full grid of arbitrary resolution but without introducing additional grid points into the calculation. This allows local nonlinear couplings to be computed quickly without the introduction of additional sample points and without the introduction of additional approximations which must be controlled.

These algorithms have been implemented in Fortran90 and have subsequently been adopted for use in electronic structure computations as described in the introduction. Prior to this, we had been using very simple, naïve $O(N^2)$ algorithms which implement each transform and operator as multiplication by the multiscale representation of the corresponding matrix. These multiplies check all points for being within interaction range and then construct the appropriate matrix element as needed. This is required in the naïbe approach because the variety of inter-scale matrix elements is too wide to store in table of reasonable size. This algorithm ultimately scales quadratically with the number of refinement levels for our application. This is because, as described in the introduction, basis functions are kept in the basis whenever they contain an atomic nucleus within their support. All functions in this subset of significant size of the basis functions associated with each atomic center therefore touch one another, and the multiscale matrices contain dense blocks connecting all of these elements of a given center with one another. Because the number of functions associated with a given center grows linearly with the number of refinement scales k, the number of operations required in the naïve approach of multiplying directly by these dense matrices scales quadratically with the number of functions in the basis. For reference, a typical number of refinement levels in electronic structure calculations of the lighter elements would be $k = 5$, as employed in the carbon atom [2] and the nitrogen molecule [1].

A comparison with the previous implementation in Fortran90 on the same processor (Superscalar SPARC Version 9, UltraSPARC) demonstrates the speed improvements and scaling which can be achieved with the new approach. The "time" axis is the CPU time taken by one application of the ∇^2 operator. The "k" axis represents the number of levels of refinement made in the basis and is proportional to the number of points in S.

Figure 9 compares the runtimes of ∇^2 in three dimensions on a 1-level touching good basis with 3rd order interpolets consisting of concentric cubes of size 15^3 centered about one atomic nucleus, as would be appropriate for the calculation of the electronic structure of a single atom. Although there is initially a significant $O(N)$ contribution, as a function of the number of refinement levels k, the times for the naïve approach show the constant increments in slope characteristic of a quadratic function. The new approach compares very favorably and is about thirty times faster for typical values of k. (Note the difference in vertical scale between the two figures.)

Although the comparison in Figure 9 is quite favorable for the new algorithm, one must bear in mind that given the typical decay in the interpolet expansion coefficients about an atom[2], [1], the functions which are appropriate to maintain in the expansions tend to have the 2-level touching property, not the 1-level touching property. Figure 10 compares the runtimes of ∇^2 in three dimensions on a 2-level touching good basis of concentric cubes of size 9^3, where the speed up just as dramatic as before, now by approximately a factor of 40.

Figure 11 compares the runtimes of ∇^2 in three dimensions on a 2-level touching good basis of two refinement centers, with refinements now consisting of cubes of size 9^3 (similar to figure 5). This situation arises in the the calculation of the electronic ground state of the nitrogen molecule, N_2. Note that with the introduction now of two atomic centers the times are again consistent with the scalings described above: The runs times only double in the multilevel algorithm but quadruple in the naïve algorithm.

Having considered the efficiency of the algorithms, we next turn to the use of these algorithms in the solution of Poisson's equation to determine electrostatic fields, which was the rate limiting step in the calculations carried out in [2] and [1]. From those calculations, we were aware that the combination of conjugate gradients with the simple preconditioning consisting of just applying the inverse of the diagonal elements of the Laplacian matrix leads to an algorithm requiring very few iterations. It was the application of the operator within each conjugate gradient iteration which limited the efficiency of those earlier calculations.

Figures 12-14 illustrate results for varying levels of refinement in the two different systems. The first system consists of refinements centered about a single atomic center within a cubic super cell of side 15 Bohr radii with periodic boundary conditions. (One Bohr radius is approximately 0.529 Angstroms.) The second system contains two refinement centers separated at a distance of 2 Bohr radii, approximately the inter-nuclear separation in the nitrogen molecule. This latter system resides within a rectangular supercell of dimensions $(15\text{ Bohr})^2 \times (17\text{ Bohr})$. In both cases, the spacing of the grid at the coarsest scale is 1 Bohr, and the finest spacing is 2^{-k} Bohr. At $k = 22$, the greatest refinement considered in our numerical experiments, the finest grid spacing is approximately 0.24×10^{-6}Å. A full grid at this resolution would contain 2.8×10^{23} points. Our truncated basis contains only about 60,000 functions in this case.

Figure 12 compares, as a function of the number of refinement levels k, the condition number of the Laplacian represented in a truncated interpolet basis (the "stiffness matrix" for the basis) and in an untruncated orthogonal basis at the corresponding resolution. The figure also shows the effect on the condition number of the interpolet stiffness matrix of the simple diagonal preconditioner described above. The condition numbers for the truncated interpolet bases were determined numerically using the operators implemented as described above. The curves indicate results for the system with a single atomic center, and the symbols indicate results for the two atom system. Comparing the results for the one and two atom cases suggests that apart from some transient behavior for small k, the condition number is not sensitive to the number of atoms and depends primarily on the number of refinement levels k.

Although finite basis representations of the Laplacian constructed from orthogonal functions at a given resolution should all have similar condition numbers, the fact that the interpolet basis is not orthogonal allows the condition numbers of multiscale interpolet operators to be quite different that their single-scale counter parts. Compared to an orthogonal basis, the condition number in the interpolet representation is already over two orders of magnitude superior at the typical $k = 5$ levels of refinement. This comparison continues to improve with increasing scale. The orthogonal condition number scales inversely as the square of the spacing on the grid of maximum resolution whereas the interpolet condition number scales inversely with approximately the 5/4 power of the resolution, as determined from the slope in the figure. The interpolet basis itself therefore provides an intrinsic form of preconditioning. Figure 12 shows that our simple explicit, diagonal preconditioner improves the scaling of the condition number, which now scales merely as the inverse of the resolution. (Note the lower slope of the lower curve.) At $k = 5$ levels of refinement the improvement is only a factor of three but becomes more significant as the number of refinements increases. We extrapolate, based upon the observed scaling behavior, that the improvement is by a factor of sixty at $k = 22$ levels of refinement, the greatest refinement considered in the examples below.

Figure 13 shows the convergence of the preconditioned conjugate gradient algorithm in the solution of Poisson's equation. As a simple example, we solve for the electrostatic potential which arises from the two nuclei in a nitrogen molecule. In this calculation we use $k = 8$ levels of refinement, a somewhat higher level of resolution than would be employed in calculations of the N_2 molecule. For this calculation, the charge density of each nucleus is modeled as a three dimensional Gaussian of root mean square width σ along each direction equal to the spacing on the finest scale. After an initial phase of about twenty iterations, the convergence becomes nearly perfectly exponential. This procedure reduces the magnitude of the residual vector by ten orders of magnitude in one hundred iterations. This is very good performance for a system consisting of 14,000 degrees of freedom with a Laplacian operator with a nominal single-scale condition number of about 65,000 at this level of resolution. The slope of this exponential portion of the convergence curve corresponds to a reduction in error at each iteration by 25%. One would obtain the same error reduction in a simple weighted iterative Jacobi algorithm (with the inverse of the maximum eigenvalue as the weight) applied to an operator with condition number $c \approx 4$. The quantity c, the inverse of the fractional improvement in the magnitude of the residual, we define as the *effective condition number* for the algorithm.

Figure 14 shows this effective condition number c for the conjugate gradient algorithm with simple diagonal preconditioning as a function of the number of refinement levels k for the solution of Poisson's equation for the nuclei in the nitrogen molecule. In all cases the extent of the nuclei σ is again set to the spacing of the finest grid. We note that after about six refinements, the effective condition number is essentially constant. The example from Figure 13 is therefore representative of the typical rate of convergence attained. These results indicate that, regardless of the amount of refinement, a constant number of iterations will suffice to produce a result of a given accuracy, even as the nominal condition number for an orthogonal stiffness at the corresponding resolution approaches 1.8×10^{13} at $k = 22$. Because the computational work involved in each iteration is linear in the number of points in the basis, this approach appears to produce the solution to Poisson's equation in O(N) time for these multiresolution bases.

A Notation

C_k	$2^k \cdot Z^n$ for $k \geq 0$, Z^n for $k < 0$.
D_k	$C_{k-1} - C_k$ for $k \geq 0$, for $k < 0$.
θ_k	$\min(k, m)$ where m is the largest integer such that 2^m divides all the components of y.
$\mathcal{F}_k(S)$	the space of functions over $S \subset Z^n$.
$\mathcal{I}_k(\phi, S)$	the space of linear combinations of $\phi(\frac{x-y}{2^{\theta_k(y)}})$ for $y \in S \subset Z^n$.
ι_k^ϕ	the mapping from $S \to \mathcal{I}_k(\phi, S)$ which takes $y \to \phi(\frac{x-y}{2^{\theta_k(y)}})$, and linearly extended to a map from $\mathcal{F}_k(S) \to \mathcal{I}_k(\phi, S)$.
$\mathcal{F}_k^S$	$\mathcal{F}_k(Z^n)/\mathcal{F}_k(Z^n - S)$.
$\mathcal{I}_k^S(\phi)$	$\mathcal{I}_k(\phi, Z^n)/\mathcal{I}_k(\phi, Z^n - S)$.
$\tilde{v}$	the zero-lift representative of $v \in \mathcal{F}_k^S$, i.e. $\tilde{v} \in \mathcal{F}_k$, such that $\tilde{v}(y) = v(y), y \in S$ and $\tilde{v}(y) = 0, y \notin S$.
V^*	the dual space of the vector space V.
J_{k_1,k_2}	the map, $\mathcal{F}_{k_1} \to \mathcal{F}_{k_2}$, given by $J_{k_1,k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$.
J_k	short for $J_{0,k}$.
J_{-k}	short for $J_{k,0}$.
$\langle f \| \mathcal{O} \| g \rangle$	the matrix element $\int f(x) \mathcal{O} g(x) d^n x$.

B Table of Contents

Contents

References

[1] T. A. Arias. Multiresolution analysis of electronic structure: semicardinal and orthogonal wavelet bases. Submitted to *Reviews of Modern Physics*, 1997.

[2] T.A. Arias, K.J. Cho, Pui Lam, and M.P. Teter. Wavelet transform representation of the electronic structure of materials. In Rajiv K. Kalia and Priya Vashishta, editors, *Proceeedings of the '94 Mardi Gras Conference: Toward Teraflop Computing and New Grand Challenge Applications*, page 23, Commack, New York, 1995. Nova Science Publishers.

[3] S. Bertoluzza and G. Naldi. A wavelet collocation method for the numerical solution of partial differential equations. *Applied Computational and Harmonic Analysis*, 3:1–9, 1996.

[4] G. Beylkin, R.R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms. *Commun. Pure and Appl. Math*, 44:141–183, 1991.

[5] G. Beylkin and J. M. Keiser. On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases. *PAM Report*, 262, 1995.

[6] G. Beylkin and J. M. Keiser. On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases. *Journal of Computational Physics*, 132:233–259, 1997.

[7] G. Beylkin and N. Saito. Wavelets, their autocorrelation functions, and multiresolution representation of signals. *Expanded abstract in Proceedings ICASSP-92*, 4:381–384, 1992.

[8] G. Beylkin and N. Saito. Multiresolution representations using the auto-correlation functions of compactly supported wavelets. *IEEE Transactions on Signal Processing*, 41:3584–3590, 1993.

[9] K. Cho, T.A. Arias, J.D. Joannopoulos, and Pui K. Lam. Wavelets in electronic structure calculations. *Physical Review Letters*, 71:1808, 1993.

[10] A. Cohen and R. Danchin. Multiscale approximation of vortex patches. *preprint*, 1997.

[11] S. Dahlke, W. Dahmen, R. Hochmuth, and R. Schneider. Stable multiscale bases and local error estimation for elliptic problems. *Applied Numerical Mathematics*, 23(1):21–47, 1997.

[12] I. Daubechies. Orthonormal bases of compactly supported wavelets. *J. Comm. Pure Appl. Math.*, 41:909–996, Oct 1988.

[13] I. Daubechies. *Ten Lectures on Wavelets*. Number 61 in CBMS/NSF Series in Applied Math. SIAM, 1992.

[14] G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constr. Approx.*, 5(1):49–68, 1989.

[15] D. L. Donoho. Interpolating wavelet tranforms. Preprint, Department of Statistics, Stanford University, 1992.

[16] S. Dubuc. Interpolation through and iterative scheme. *J. Math. Anal. Appl.*, 114:185–204, 1986.

[17] J. Frohlich and K. Schneider. An adaptive wavelet galerkin algorithm for one- and two-dimensional flames. *Computations. Eur. J. Mech*, 13:439–471, 1994.

[18] S. Goedecker and O.V. Ivanov. Linear scaling solution of the coulomb problem using wavelets. *Preprint (see* `http://xxx.lanl.gov/abs/physics/9701024`*)*, Submitted Jan 1997.

[19] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136:B864–B871, 1964.

[20] W. Kohn and L.J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140:A1133, 1965.

[21] J. Liandrat, V. Pierrier, and Ph. Tchamitchian. Numerical resolution of the regularized Burgers equation using the wavelet transform. *Tech. Report CPT-89/P. 2320 - Center of Theoretical Physics, Marseille, France*, 1989.

[22] J. Liandrat and P. Tchamitchian. Resolution of the 1d regularized burgers equation using a spatial wavelet approximation. report, Technical Report 90-83, NASA ICASE, December 1990.

[23] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, and J.D. Joannopoulos. Iterative minimization techniques for *ab initio* total energy calculations: molecular dynamics and conjugate gradients. *Reviews of Modern Physics*, 64:1045, 1992.

[24] N. Saito and G. Beylkin. Multiresolution representations. *Applied and Computational Harmonic Analysis*, 41:3584, 1993.

[25] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Cambridge, Massachusetts, 1996.

[26] C.J. Tymczak and Xiao-Qian Wang. Orthonormal wavelet bases for quantum molecular dynamics. *Physical Review Letters*, 78:3654, 1997.

[27] Siqing Wei and M.Y Chou. Wavelets in self-consistent electronic structure calculations. *Physical Review Letters*, 76:2650, 1996.

[28] Dicle Yeşilleten and T.A. Arias. Synthesis of wavelet theory with the multigrid algorithm. Condensed Matter Physics, MIT, 1996.

Figures: (available from http://laisla.mit.edu/muchomas/Papers/nonstand-figs.ps or in the file "figs.ps" in this posting)

Figure 1: (a) shows $\phi(x/2) = \frac{-1}{16}\phi(x-3) + \frac{9}{16}\phi(x-1) + \phi(x) + \frac{9}{16}\phi(x+1) + \frac{-1}{16}\phi(x+3)$. (b) shows the functions $\frac{-1}{16}\phi(x-3)$, $\frac{9}{16}\phi(x-1)$, $\phi(x)$, $\frac{9}{16}\phi(x+1)$, $\frac{-1}{16}\phi(x+3)$.

Figure 2: (a) the smooth function $f(x) = (\frac{x}{2}+3)^3 e^{-(\frac{x}{4})^4}$. (b) the approximation to $f(x)$ in $\mathcal{I}_0(Z)$. (c) component of the approximation in $\mathcal{I}_1(C_1)$. (d) component of the approximation in $\mathcal{I}_1(D_1)$.

Figure 3: Approximation of the function $f(x) = e^{-|x|}$ with interpolets centered at the tick marks: (a) a graph of $e^{-|x|}$. (b) the cardinal approximation constructed from interpolets centered at the knots of a uniform grid. (c) and (d) cardinal approximations from local refinements of the uniform grid. The additional levels of refinement near the singularity increase the accuracy.

Figure 4: (left) a graph of the $\mathcal{L}^2$ relative error for k levels of local dyadic refinement. (right) a graph of the $\mathcal{L}^\infty$ relative error for k levels of local dyadic refinement.

Figure 5: An example of a truncation one might wish to use for a diatomic molecule (two atomic cores). In black are shown those points in $S \subset Z^n$ whose residual values might be significant.

Figure 6: A visual summary of the 1-level touching condition. Solid plots represent functions centered at points in S. Dotted plots represent functions centered at points in $Z^n - S$. Tick marks delimit the overlap of the two functions.

Figure 7: Our example of the 1-level touching good basis in one dimension. Note that the two functions plotted do not touch.

Figure 8: A generic example of a truncation which meets our definitions of good and 1-level touching.

Figure 9: The previously used implementation is on the left, and the implementation employing a 1-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)

Figure 10: The previously used implementation is on the left, and the implementation employing a 2-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)

Figure 11: The previously used implementation is on the left, and the implementation employing a multilevel algorithm is on the right. (Note the difference in scale on the vertical axes.)

Figure 12: The condition number of the Laplacian operator represented in truncated multiresolution interpolet basis as a function of the number of refinement levels k with and without simple diagonal preconditioning and compared with the condition number in an orthogonal basis with the same resolution. Lines indicate results for bases with a single atomic center of refinement and points represent results for two atomic centers corresponding to the nitrogen molecule.

Figure 13: Convergence of the solution to Poisson's equation for the nuclear potential in a nitrogen molecule in an interpolet basis with $k = 8$ levels of refinement about each nucleus.

Figure 14: Effective condition number of Poisson's equation for the nuclear potential in a nitrogen molecule with simple diagonal preconditioning as a function of k the number of levels of refinement about each nucleus.