SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. S123–S142

MADNESS: A MULTIRESOLUTION, ADAPTIVE NUMERICAL ENVIRONMENT FOR SCIENTIFIC SIMULATION*

ROBERT J. HARRISON†, GREGORY BEYLKIN‡, FLORIAN A. BISCHOFF§, JUSTUS A. CALVIN¶, GEORGE I. FANN‖, JACOB FOSSO-TANDE#, DIEGO GALINDO‖, JEFF R. HAMMOND**, REBECCA HARTMAN-BAKER††, JUDITH C. HILL‖, JUN JIA‡‡, JAKOB S. KOTTMANN§, M-J. YVONNE OU§§, JUNCHEN PEI¶¶, LAURA E. RATCLIFF‖‖, MATTHEW G. REUTER†, ADAM C. RICHIE-HALFORD##, NICHOLS A. ROMERO‖‖, HIDEO SEKINO†††, WILLIAM A. SHELTON‡‡‡, BRYAN E. SUNDAHL†, W. SCOTT THORNTON†, EDWARD F. VALEEV¶, ÁLVARO VÁZQUEZ-MAYAGOITIA‖‖, NICHOLAS VENCE§§§, TAKESHI YANAI¶¶¶, AND YUKINA YOKOI†††

Abstract. MADNESS (multiresolution adaptive numerical environment for scientific simulation) is a high-level software environment for solving integral and differential equations in many dimensions that uses adaptive and fast harmonic analysis methods with guaranteed precision that are based on multiresolution analysis and separated representations. Underpinning the numerical

*Received by the editors June 16, 2015; accepted for publication (in revised form) April 6, 2016; published electronically October 27, 2016. This research was sponsored in part by the Office of Advanced Scientific Computing Research's Math Program and the SciDAC Program, U.S. Department of Energy. Some work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle under contract DE-AC05-00OR22725. This work also employed resources of the National Center for Computational Sciences at Oak Ridge National Laboratory and of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357.

http://www.siam.org/journals/sisc/38-5/M102617.html

†Corresponding author. Stony Brook University, Stony Brook, NY 11794 (robert.harrison@stonybrook.edu, matthew.reuter@stonybrook.edu, bryan.sundahl@stonybrook.edu, william.thornton@stonybrook.edu). The first author's work was supported in part by the National Science Foundation under grant ACI-1450344.

‡University of Colorado at Boulder, Boulder, CO 80309 (gregory.beylkin@colorado.edu).

§Institut für Chemie, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany (florian.bischoff@chemie.hu-berlin.de, jakob.kottmann@chemie.hu-berlin.de).

¶Department of Chemistry, Virginia Tech, Blacksburg, VA 24061 (justusc@vt.edu, evaleev@vt.edu). The work of these authors was supported in part by the National Science Foundation under grants ACI-1047696 and ACI-1450262.

‖Oak Ridge National Laboratory, Oak Ridge, TN 37831 (fanngi@ornl.gov, galindo_diego_a@cat.com, hilljc@ornl.gov).

#Department of Chemistry and Biochemistry, Florida State University, Tallahassee, FL 32306 (fossotaj@gmail.com).

**Parallel Computing Lab, Intel Corporation, Portland, OR 97219 (jeff.science@gmail.com).

††National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (hartmanbaker@gmail.com).

‡‡LinkedIn, Mountain View, CA 94043 (jakiej@gmail.com).

§§Department of Mathematical Sciences, University of Delaware, Newark, DE 19716 (miaok@gmail.com).

¶¶State Key Laboratory of Nuclear Physics and Technology, School of Physics, Peking University, Beijing 100871, China (cpei.pku@gmail.com).

‖‖Argonne Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439 (lratcliff@anl.gov, naromero@gmail.com, fray.gory@gmail.com).

##Department of Physics, University of Washington, Seattle, WA 98195 (richiehalford@gmail.com).

†††Computer Science and Engineering, Toyohashi University of Technology, Toyohashi, AICHI 441-8580, Japan (sekinoh@gmail.com, yokoiykn@gmail.com).

‡‡‡Louisiana State University, Baton Rouge, LA 70803 (sheltonwair@gmail.com).

§§§Department of Physics, LaSierra University, Riverside, CA 92505 (nickvence@gmail.com).

¶¶¶Theoretical and Computational Molecular Science, Institute for Molecular Science, Okazaki, AICHI, 444-8585, Japan (yanait@ims.ac.jp).

capabilities is a powerful petascale parallel programming environment that aims to increase both programmer productivity and code scalability. This paper describes the features and capabilities of MADNESS and briefly discusses some current applications in chemistry and several areas of physics.

Key words. scientific simulation, multiresolution analysis, high-performance computing

AMS subject classifications. 65-01, 65Y05, 65Z05, 65E05, 65N99, 65T60, 00A72, 81-08

DOI. 10.1137/15M1026171

1. Introduction. Numerical tools are ubiquitous in modern science since the relevant models/equations do not generally have analytical solutions. Although advances in scientific computing over the last thirty years have enabled more sophisticated models and the quantitative simulation of large-scale problems, these numerical methods introduce new, unphysical considerations that must also be taken into account. For instance, function representation, that is, the choice of basis set, is critically important. If a basis set does not adequately resolve the finest details of the system, the calculated solutions will likely be inaccurate and have questionable physical interpretation. Other numerical factors include computational efficiency, scalability, distributed memory management, differentiation schemes, and quadrature rules. As such, most computational approaches force the scientist to compose software in terms of details inherent to the representation (e.g., the coefficients of basis functions or integrals within the basis set), rather than at the level of mathematics or physics that uses the natural functions or operators of the problem domain. In the end, all of these considerations obscure the desired science and force scientists to instead focus on low-level computational details.

Several software packages have been developed to help insulate scientists from these issues; notable examples are Trilinos [31], PETSc [9], FEniCS [40], and Chebfun [22]. In essence, these and similar frameworks support common scientific computing operations, e.g., linear algebra algorithms, mesh management (function representation), nonlinear solvers, and ordinary differential equation integrators. These packages also typically scale to parallel or massively parallel machines, facilitating the solution of large-scale problems. Most of these packages, however, require the scientist to "think" in vectors and matrices instead of functions and operators (although some packages, such as FEniCS, provide additional capabilities). Furthermore, computation typically focuses on three or fewer dimensions.

Our goal in this communication is to motivate and overview the MADNESS (multiresolution adaptive numerical environment for scientific simulation) package, which offers the following capabilities to users:

1. The illusion of basis-free computation. Under this illusion, the user computes with functions and operators instead of vectors and matrices. Behind the scenes, functions and operators are expanded in a multiwavelet basis, where the number of basis functions is adapted to guarantee the precision of each operation. As necessary, the "mesh," which is the support of the basis functions, is refined in computationally-troublesome regions (perhaps those with fine details) and coarsened in others, enabling the simulation of multiscale problems.

2. Fast and arbitrarily accurate methods for solving differential and integral equations in one to six dimensions (perhaps more), with specified boundary conditions. These operations include, for example, linear algebra, numerical differentiation/integration, and, perhaps most importantly, integral convolution kernels, such as the Poisson equation Green's function. In some applications of MADNESS, for

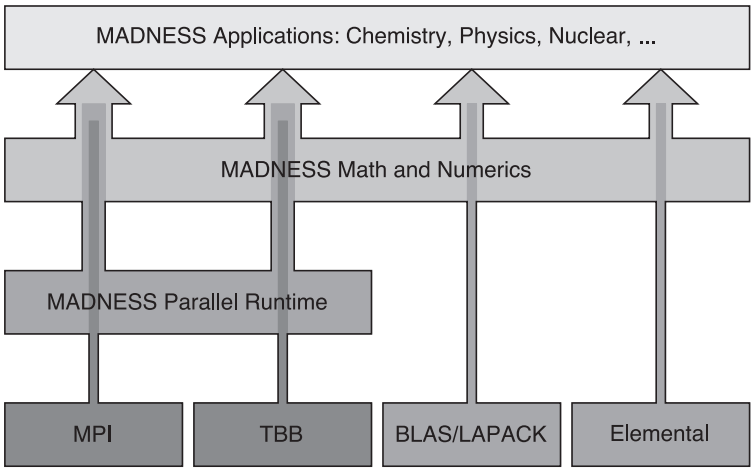

FIG. 1. *The library structure of MADNESS and its dependencies. Each of these sub components has its own public application programing interface that may be used both for development inside MADNESS as well as in external software packages.*

instance, many-body quantum mechanics [21], such high-dimensional functions and operators are essential.

3. A parallel runtime that allows the user to compose algorithms as a dynamically scheduled set of tasks that operate on objects in global namespaces.

4. Algorithms and implementations that scale to massively parallel computational resources, thereby enabling the solution of large problems.

Figure 1 schematically describes these capabilities and how they interoperate with each other (and with external libraries) to provide the functionalities of MADNESS to users.

We aim, by providing these tools, to raise the level of composition of scientific applications, making it faster and easier to both (i) construct robust and correct algorithms and (ii) calculate solutions to new and existing problems. Although MADNESS's breakthrough initial application was in computational chemistry [29, 56, 57], its framework more generally represents functions, operators, and boundary conditions. Thus, it is readily able to describe and solve other well-defined problems. To date, applications include atomic and molecular physics, electrostatics, fluid dynamics, graph theory, materials science, nanoscience, nuclear physics, and solvation models [24, 25, 34, 44, 45, 51, 54]. We hope that this introduction to MADNESS conveys its generality and encourages applications in new domains.

The layout of this communication is as follows. Section 2 introduces the basics of multiresolution analysis (MRA), which is the mathematical underpinning of MADNESS, and how MRA facilitates scientific computing. We then provide and discuss, in section 3, some simple examples of programming in MADNESS that illustrate the absence of basis set considerations and the ability to write code in terms of functions and operators. Next, section 4 summarizes the computational details of MADNESS, including the runtime environment/framework and the methods behind MADNESS's utilization of (massively) parallel architectures. Finally, section 5 summarizes this report and describes future directions for MADNESS.

2. Numerics: Mathematical overview & capabilities. With the illusion of basis-free computation, a MADNESS user simply needs to specify the desired accu-

racy; from this, the numerical framework provides fast computation with guaranteed accuracy (in up to six dimensions). This is accomplished through several key mechanisms:

1. The underlying representation is an orthonormal basis [7]; functions are represented on an adaptive mesh with a discontinuous spectral-element basis set. Within each adaptively refined element the basis functions are tensor products of Legendre polynomials. Crucially, the application programmer only uses the function as a single entity and never sees or manipulates the basis or mesh.

2. Guaranteed accuracy for each operation (e.g., multiplication or addition of functions, application of an operator) is accomplished through automatic refinement of the mesh using criteria derived from MRA [7].

3. Fast and accurate computation is enabled through MRA by separating behavior between length scales. MRA provides a sparse hierarchical (tree) decomposition of functions and operators that allows automatic refinement or coarsening of the mesh to meet the accuracy goal. MRA also yields fast algorithms for physically important operators [7, 12, 23].

4. Fast and accurate computation beyond one dimension is made feasible through accurate and compact representations of operators as sums of separable kernels (usually Gaussians), which are created through quadrature or more advanced numerical techniques [12, 13, 28].

In the remainder of this section, we elaborate on these points. Novel contributions first demonstrated within MADNESS include dynamic construction and application of low-separation rank approximations [12, 28, 29] for the Poisson and bound-state Helmholtz integral kernels, which, in combination with use of the nonstandard form [10], enabled the first practical application of multiresolution techniques in three and higher dimensions.

2.1. Underlying basis and adaptive mesh. Inside MADNESS (using a one-dimensional example here for simplicity), the user's computational domain is mapped to $[0,1]$. This domain is repeatedly subdivided by factors of two such that, at level n, there are 2^n boxes, each of size 2^{-n}. Within each box, we use the first k Legendre polynomials as the basis (or scaling functions in the language of MRA [6, 7]). The polynomials are scaled and shifted to form an orthonormal basis.

Specifically, the ith scaling function ($i = 0, \ldots, k-1$) in the lth box ($l = 0, \ldots, 2^n - 1$) at level n ($n = 0, 1, \ldots$) is

$$\phi_{li}^n(x) = 2^{n/2}\phi_i(2^n x - l),$$

where

$$\phi_i(x) = \begin{cases} \sqrt{2i+1}\,P_i(2x-1), & x \in (0,1), \\ 0, & x \notin (0,1), \end{cases} \tag{2.1}$$

is the ith mother scaling function. Projection of a function $f(x)$ into the basis at level n is accomplished by orthogonal projection,

$$f^n(x) = \sum_l \sum_i s_{li}^n \phi_{li}^n(x), \tag{2.2a}$$

$$s_{li}^n = \int \mathrm{d}x f(x)\phi_{li}^n(x). \tag{2.2b}$$

The error in this approximation is given (in one dimension) by Lemma 1.1 in [6],

$$\|f^n - f\| \leq \frac{2^{1-(n+2)k}}{n!} \sup_{x\in[0,1]} \left| \frac{\mathrm{d}^n}{\mathrm{d}x^n} f(x) \right|. \tag{2.3}$$

Projection is not performed globally on a uniform fine grid; instead, projection starts on a relatively coarse level (n). Within each box on level n we compute the projection at level n and at the coarser level $n-1$. If the difference between these projections is not satisfactory, the projection is locally repeated at a finer level ($n+1$), and so forth, until the projection is sufficiently accurate.

Subsequent to this projection, operations on functions may trigger coarsening or refinement to meet accuracy goals [7]. For instance, pointwise multiplication of two functions can trigger local refinement by one level if accurate representation of the product requires polynomials exceeding degree $k-1$. Another example of refinement comes when applying integral operators. Contributions from a source box to increasingly distant neighboring boxes must be sampled until the result is negligible, with the required distance depending on both details within the source box and the operator being applied. Because of these considerations, each function in a computation (and applications in nuclear and electronic structure may employ thousands of functions) has its own independent mesh.

2.2. Multiresolution analysis. Rather than thinking about the approximation of a function at each length scale, MRA focuses upon the difference between length scales. In this way, MRA exploits both smoothness and sparsity for fast computation. The telescoping series illustrates this. Let V^n be the space spanned by the polynomials at level n. Then, we can write

$$V^n = V^0 \oplus \left(V^1 \ominus V^0\right) \oplus \left(V^2 \ominus V^1\right) \oplus \cdots \oplus \left(V^n \ominus V^{n-1}\right), \tag{2.4}$$

which decomposes the approximation space at level n in terms of the coarsest approximation (V^0) plus corrections at successive levels of refinement. Note that $\oplus$ is the direct sum and $V^n \ominus V^{n-1}$ denotes the orthogonal complement of V^{n-1} in V^n. If the function of interest is sufficiently smooth in some volume, then differences at finer scales will be negligible in that volume. Thus, smoothness is translated into sparsity and hence fast computation; a precise definition of "negligible" enables adaptive refinement to guarantee accuracy. The multiwavelets at level n are defined as an orthonormal, disjoint basis that spans the difference space $W^n = V^{n+1} \ominus V^n$ and are constructed by translation and dilation of the mother wavelets [6, 7].

We thus have two representations of our function: (i) its projection onto the finest level of the adaptively refined grid (i.e., in real space or in the scaling function basis) and (ii) its representation at the coarsest level plus corrections at successive levels of refinement (i.e., in the multiwavelet basis). The fast wavelet transform [11] enables fast and accurate transformation between these two representations. From a practical perspective, we simply pick the most convenient basis to compute in, just as one would if using the Fourier basis and the fast Fourier transform. However, the disjoint support of multiwavelets enables local adaptive refinement, which is not possible in the global Fourier basis. A third equivalent representation is the value of the function at the Gauss–Legendre quadrature points within each box of the mesh, which facilitates certain operations (such as multiplication of functions).

We can use a similar approach to understand how to turn smoothness into sparsity when applying operators. Let P^n and Q^n be the projectors onto V^n and W^n,

respectively. Then, by construction, $P^{n+1} = P^n + Q^n$. Let T be the integral kernel of an operator we wish to apply, and let us assume that level n is adequate to resolve both the input and output functions. The representation of the operator at level n is

$$\begin{aligned} T^n &= P^n T P^n \\ &= \left(P^{n-1} + Q^{n-1}\right) T \left(P^{n-1} + Q^{n-1}\right) \\ &= T^{n-1} + Q^{n-1} T Q^{n-1} + Q^{n-1} T P^{n-1} + P^{n-1} T Q^{n-1} \\ &= T^0 + \sum_{m=0}^{n-1} \left(A^m + B^m + C^m\right), \end{aligned} \tag{2.5}$$

where we define $A^m = Q^m T Q^m$, $B^m = Q^m T P^m$, and $C^m = P^m T Q^m$. In this so-called nonstandard form [10], T^0 acts only upon scaling functions at the coarsest level, whereas the operators applied at finer levels all involve wavelets.

For a demonstrative example of the computational significance of this approach, let us examine convolution with the Poisson integral kernel [7, 28, 29], $T = |\vec{x} - \vec{y}|^{-1}$, which is employed to compute the electrostatic potential arising from a charge distribution. (See the example in section 3.2.) Consider an element of the matrix A^m, which corresponds to the electrostatic interaction between a wavelet in a box at level m and a wavelet in another box at the same level. By construction, the first k moments of the wavelets vanish. (This is a consequence of wavelets being orthogonal to scaling functions at the same or coarser scales, expressed above in the definition $W^n = V^{n+1} \ominus V^n$.) Thus, the first $2k$ terms in the double Taylor series expansion of the interaction potential T are zero (because of the orthogonality in both x and y). Consequently, and as further discussed in Appendix A, we see that the matrix elements of A^m decay as $|\vec{x} - \vec{y}|^{-2k-1}$. For instance, if $k = 8$, which is routine, they decay as $|\vec{x} - \vec{y}|^{-17}$. In other words, even though T is dense, A^m is very sparse to any finite precision. A similar approach shows that B^m and C^m decay as $|\vec{x} - \vec{y}|^{-k-1}$. Translationally invariant operators yield Toeplitz representations, which are very important for practical computation due to the greatly reduced burden of computing and storing the operator.

2.3. Separated representations. The approach described above is not efficient beyond one dimension because the cost of naïvely applying an operator grows as $O\left((2k)^{2D} \log(\epsilon^{-1})^D\right)$ with similarly excessive memory requirements, where D is the dimensionality and ϵ is the desired precision. However, for many physically important operators we can construct [12, 28, 29] either global or local (via numerically generated low-rank factorizations of the full operator) separated representations. In other words, we can approximate, with controlled precision, the full operator as a sum of products of operators that separately apply in each dimension. As we will see shortly, the number of terms in the sum will be small. This reduces the cost of application to $O\left(NM(2k)^{D+1} \log(\epsilon^{-1})^D\right)$ and the memory footprint to $O\left(MD(2k)^2 \log(\epsilon^{-1})\right)$, where M is the number of terms in the sum and N is the size of basis for the function to which the operator is being applied. Powerful techniques for generating these separated forms include quadrature of integral identities [28] and numerical techniques developed by Beylkin and Monzón [13]. For example, numerical quadrature to evaluate the integral

$$r^{-1} = \frac{2}{\sqrt{\pi}} \int_{-\infty}^{\infty} \mathrm{d}s e^{-r^2 e^{2s} + s}$$

yields compact representations of r^{-1} as a sum over Gaussians,

$$= \sum_{\mu=1}^{M} c_\mu e^{-t_\mu r^2} + O\left(\epsilon r^{-1}\right), \tag{2.6}$$

in which the number of terms (M) scales only as $O((\log R)(\log \epsilon^{-1}))$, where $[1, R]$ is the range of validity and ϵ is the relative precision.

3. Applications: Programming with MADNESS. Having discussed the mathematical underpinnings of MADNESS, we now examine some samples of numerical programming with MADNESS. Many additional examples, including quantum chemistry and other applications, are provided in the MADNESS code repository. (See the directories `src/apps/` and `src/examples/` in the code [1].)

3.1. Example: Getting started with MADNESS. Our first example aims to evaluate the integral (trace), 2-norm, and electrostatic self-energy of a Gaussian function in three dimensions:

$$g(\vec{x}) = e^{-|\vec{x}|^2}, \tag{3.1a}$$

$$\int \mathrm{d}^3\vec{x}\; g(\vec{x}) = \pi^{3/2} \doteq 5.5683279, \tag{3.1b}$$

$$\left(\int \mathrm{d}^3\vec{x}\; g(\vec{x})^2\right)^{1/2} = (\pi/2)^{3/4} \doteq 1.403104, \tag{3.1c}$$

$$\int \mathrm{d}^3\vec{x}\; g(\vec{x}) \int \mathrm{d}^3\vec{y}\; |\vec{x} - \vec{y}|^{-1} g(\vec{y}) = \pi^{5/2} 2^{1/2} \doteq 24.739429. \tag{3.1d}$$

Listing 1 shows the entire MADNESS source code for this task, which we now walk through.

```
1  #include <madness/mra/mra.h>
2  #include <madness/mra/operator.h>
3  using namespace madness;
4
5  double gaussian(const coord_3d& r) {
6    double x=r[0], y=r[1], z=r[2];
7    return exp(-(x*x + y*y + z*z));
8  }
9
10 int main(int argc, char** argv) {
11   initialize(argc, argv);
12   World world(SafeMPI::COMM_WORLD);
13   startup(world,argc,argv);
14
15   FunctionDefaults<3>::set_cubic_cell(-6.0,6.0);
16   real_convolution_3d op = CoulombOperator(world, 1e-4, 1e-6);
17   real_function_3d g = real_factory_3d(world).f(gaussian);
18   print(g.trace(), g.norm2(), inner(g,op(g)));
19
20   finalize();
21   return 0;
22 }
```

Listing 1

Code for evaluating the integrals in equation (3.1)*; comments have been removed for brevity. This code can be found in* `src/examples/gaussian.cc`.

Initially ignoring the boiler-plate code, there are only four lines of significance:

- Line 15 defines the computational volume ($[-6, 6]^3$).
- Line 16 constructs a separated Gaussian representation of the Green's function for Poisson's equation (i.e., the Coulomb operator) with free-space boundary conditions, as described in section 2.3. The two numerical values in the constructor indicate the finest length scale to resolve and the desired accuracy of the operator, respectively.
- Line 17 projects the analytic function (lines 5–8) into the adaptive numerical basis using the default precision ($\epsilon = 10^{-4}$) and wavelet order ($k = 6$).
- Line 18 prints the results.

The integral in (3.1d) (the last printed value) is computed as the inner product (`inner`) of the Gaussian function (`g`) and its convolution with the Green's function (`op(g)`). Printed results agree with the exact values to six significant figures, which is more than might be expected from the default precision of 10^{-4}. This is a common observation and is likely due to the projection into the numerical basis oversampling to guarantee the requested precision.

The main program commences (line 11) by initializing the MADNESS parallel computing environment (discussed in section 4), which includes initializing MPI (if it has not yet been initialized). Next, line 12 creates a `World` object that encapsulates an MPI intracommunicator. This specifies the context of the distributed computation (just like an MPI communicator) but provides the much more powerful capabilities of the MADNESS parallel runtime, such as the active messages, task scheduling, and more (see section 4). Subsequently, line 13 initializes the MADNESS numerical environment. The program concludes by terminating the parallel computing environment (line 20), which also finalizes MPI if the call to `initialize()` initialized MPI.

Finally, the MADNESS framework provides support for common mathematical and computational processes. For example, algorithms are implemented for (i) matrix-free iterative solvers (such as GMRES [46] and BiCGStab [53]), (ii) converting between the multiwavelet basis and a Fourier basis [33], (iii) solving nonlinear systems of equations [27], and (iv) outputting MADNESS functions for visualization with various tools, including VisIt [4], OpenDX [2], and ParaView [30].

3.2. Example: Dielectric media. As a more substantial example, we use MADNESS to solve Poisson's equation within an inhomogeneous medium, i.e., a medium in which the permittivity is not constant,

$$\nabla \cdot \left[\epsilon(\vec{r})\nabla u(\vec{r})\right] = -4\pi\rho(\vec{r}).$$

Expanding and rearranging yields

$$\nabla^2 u(\vec{r}) = -\frac{1}{\epsilon(\vec{r})}\left[4\pi\rho(\vec{r}) + \nabla\epsilon(\vec{r}) \cdot \nabla u(\vec{r})\right]. \tag{3.2}$$

We interpret ρ/ϵ as the effective free charge density and $\nabla\epsilon \cdot \nabla u/(4\pi\epsilon)$ as the induced surface charge density. With free-space boundary conditions at long range we can invert the Laplacian by convolving with the free-space Green's function, $G(\vec{r}, \vec{s}) = -(4\pi|\vec{r} - \vec{s}|)^{-1}$. Note that MADNESS provides convolution with $1/|\vec{r} - \vec{s}|$; the user must keep track of the $-(4\pi)^{-1}$ factor. Equation (3.2), which is deliberately written in the form of a fixed point iteration, thus becomes

$$u = G * (\rho_{\text{vol}} + \rho_{\text{surf}}), \tag{3.3a}$$

where

$$\rho_{\text{vol}} = \frac{\rho}{\epsilon} \tag{3.3b}$$

and

$$\rho_{\text{surf}} = \frac{\nabla\epsilon \cdot \nabla u}{4\pi\epsilon}. \tag{3.3c}$$

We now demonstrate composition and numerical convergence of the solution to equation (3.3) within MADNESS by choosing a simple form for $\epsilon(\vec{r})$ and comparing our numerical results to the analytical solution. Specifically, consider a unit point charge placed at the center of a sphere with radius $R = 2$ (and assumed to be centered at the origin). Inside the sphere the permittivity is $\epsilon_{\text{int}} = 1$, and outside it is $\epsilon_{\text{ext}} = 10$. The exact solution of (3.2) is

$$u_{\text{exact}}(\vec{r}) = \begin{cases} \dfrac{1}{\epsilon_{\text{ext}}|\vec{r}|}, & |\vec{r}| > R, \\ \dfrac{\text{erf}(\sqrt{\xi}|\vec{r}|)}{\epsilon_{\text{int}}|\vec{r}|} + \left(\dfrac{1}{\epsilon_{\text{ext}}} - \dfrac{1}{\epsilon_{\text{int}}}\right)\dfrac{1}{R}, & |\vec{r}| < R, \end{cases} \tag{3.4}$$

where we have approximated the point charge by a normalized Gaussian with a large exponent,

$$\rho(\vec{r}) = \left(\frac{\xi}{\pi}\right)^{3/2} e^{-\xi|\vec{r}|^2}. \tag{3.5}$$

With $\xi = 100$ in what follows, this Gaussian appears as a point charge on the scale of interest. Note that $\xi = \mathcal{O}(10^9)$ may be used for a physically motivated, finite-size approximation of the charge distribution of a proton.

To implement the problem in MADNESS, we employ a diffuse-boundary approximation [44] and define a characteristic function (or mask),

$$C(\vec{r};\sigma) = H\left(\frac{|\vec{r}| - R}{\sigma}\right), \tag{3.6}$$

that is one inside the sphere and zero outside it. $|\vec{r}| - R$ is the signed distance of the point $\vec{r}$ from the surface of the sphere and $H(s) = \text{erfc}(s)/2$ is a smooth approximation to the Heaviside step function. There is a layer of width $\mathcal{O}(\sigma)$ around the surface where $C(\vec{r};\sigma)$ smoothly varies from 0 to 1. The dielectric function is then approximated as

$$\epsilon(\vec{r},\sigma) = \epsilon_{\text{ext}}\left(\frac{\epsilon_{\text{int}}}{\epsilon_{\text{ext}}}\right)^{C(\vec{r};\sigma)}. \tag{3.7}$$

This exponential form has the advantages (compared to, for instance, a linear combination of the characteristic function and its complement) that the log-derivative of the dielectric is a Gaussian located precisely at the boundary and that more rapid convergence is obtained as σ is reduced.

Listing 2 displays the MADNESS code for solving (3.3) with the above approximations. Let us discuss the parts of this code.

```
1 #include <madness/mra/mra.h>
2 #include <madness/constants.h>
3 #include "nonlinsol.h"
4 using namespace madness;
5 using namespace std;
6
```

```
7  const double eps_int = 1.0; // Interior dielectric
8  const double eps_ext =10.0; // Exterior dielectric
9  const double R = 2.0;       // Radius of sphere
10 const double xi = 100.0;    // Exponent for delta function approx.
11 const double sigma = 0.1;   // Surface "width"
12
13 class DSphere : public FunctionFunctorInterface<double, 3> {
14   const double fac;
15   const int axis;
16
17 protected:
18   double dmask(double s) const {
19     if (fabs(s) > 6.0) return 0.0;
20     return -exp(-s*s) / sqrt(constants::pi);
21   }
22
23 public:
24   DSphere(double Vint, double Vext, int axis)
25     : fac(log(Vint/Vext)), axis(axis) {}
26
27   double operator()(const coord_3d &r) const {
28     double d = sqrt(r[0]*r[0] + r[1]*r[1] + r[2]*r[2]);
29     double sdf = (d-R) / sigma;
30     return fac*r[axis]*dmask(sdf)/(sigma*d);
31   }
32 };
33
34 double rho_function(const coord_3d &r) {
35   const double coeff = pow(xi / constants::pi, 1.5);
36   return coeff*exp(-xi * (r[0]*r[0] + r[1]*r[1] + r[2]*r[2]));
37 }
38
39 int main(int argc, char **argv) {
40   const double rfourpi = 1.0/(4.0*constants::pi);
41   initialize(argc, argv);
42   World world(SafeMPI::COMM_WORLD);
43   startup(world, argc, argv);
44
45   // Function defaults
46   FunctionDefaults<3>::set_k(6);
47   FunctionDefaults<3>::set_thresh(1e-4);
48   FunctionDefaults<3>::set_cubic_cell(-5, 5);
49   FunctionDefaults<3>::set_initial_level(3);
50   FunctionDefaults<3>::set_truncate_on_project(true);
51
52   // Make integral and derivative operators
53   real_convolution_3d op = CoulombOperator(world, 1e-4,
         FunctionDefaults<3>::get_thresh());
54   real_derivative_3d Dx = free_space_derivative<double,3>(world, 0);
55   real_derivative_3d Dy = free_space_derivative<double,3>(world, 1);
56   real_derivative_3d Dz = free_space_derivative<double,3>(world, 2);
57
58   // Make functors for dielectric related quantities
59   real_functor_3d epsx_functor(new DSphere(eps_int, eps_ext, 0));
60   real_functor_3d epsy_functor(new DSphere(eps_int, eps_ext, 1));
61   real_functor_3d epsz_functor(new DSphere(eps_int, eps_ext, 2));
62
63   // Make the actual numerical functions
64   real_function_3d rho = real_factory_3d(world).f(rho_function);
65   real_function_3d eps_x = real_factory_3d(world).functor(
         epsx_functor);
66   real_function_3d eps_y = real_factory_3d(world).functor(
```

```
        epsy_functor);
67    real_function_3d eps_z = real_factory_3d(world).functor(
        epsz_functor);
68
69    real_function_3d u = op(rho).truncate(); // Initial guess
70    NonlinearSolver solver;
71    for (int iter=0; iter<20; iter++) {
72      real_function_3d surf_rho = rfourpi*(eps_x*Dx(u) + eps_y*Dy(u) +
          eps_z*Dz(u));
73      real_function_3d r = (u - op(rho + surf_rho)).truncate(); //
          residual
74      real_function_3d unew = solver.update(u, r);
75      double change = (unew - u).norm2();
76      u = unew;
77
78      print("iter", iter, "change", change, "surf charge", surf_rho.
          trace());
79
80      if (change < 10.*FunctionDefaults<3>::get_thresh()) break;
81    }
82
83    coord_3d lo{0., 0., 0.}, hi{5., 0., 0.}; // Range for line plotting
84    plot_line("testpot.dat", 501, lo, hi, u);
85
86    finalize();
87    return 0;
88  }
```

LISTING 2

Code for the dielectric media example in section 3.2; *most comments have been removed for brevity (and due to the discussion in section* 3.2*). The complete code can be found in* `src/examples/siam-example.cc`.

- Lines 1–5, 40–50, and 86–87 are the boilerplate code that initializes and finalizes MADNESS.
- Similar to the example in section 3.1, lines 34–37 provide a C++ function for creating a MADNESS representations of $\rho(\vec{r})$. Line 64 uses this C++ function to actually construct the MADNESS functions for $\rho(\vec{r})$.
- As an alternative to such a simple C++ function, lines 13–32 demonstrate the use of functors for creating MADNESS functions. Functors are necessary when the function to be projected depends on more information than just the coordinate. In our example here, the `DSphere` functor is used for projecting components of $\nabla\epsilon/\epsilon = \nabla \log \epsilon$; we need to know which component (`axis`) to project and also the radius of the sphere, etc. For simplicity, R, ξ, and σ are hard-coded.

  Going into a little more detail, lines 18–21 implement $\mathrm{d}H/\mathrm{d}s(s)$, where $s \equiv (|\vec{r}| - R)/\sigma$. The value of the functor $([\nabla \log \epsilon](\vec{r}))$ is computed at lines 27–31. In the main program, lines 59–61 create the functors. (Note the `0`, `1`, or `2` arguments for the x, y, or z directions, respectively.) Finally, lines 65–67 produce MADNESS representations of the components of $\nabla \log \epsilon$, where the functor's `()` operator takes the place of the simple C++ functions used previously.
- Line 53 initializes the Green's function operator [up to the $-(4\pi)^{-1}$ factor], and lines 54–56 create differential operators for the respective directions using the default, free-space boundary conditions.

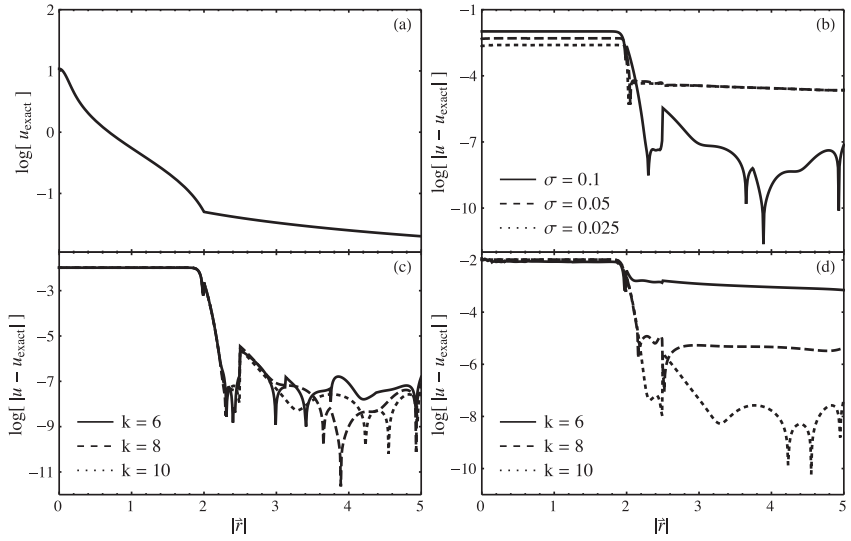

FIG. 2. *Numerical results for the example problem described in section* 3.2. (a) *The analytical result,* u_{exact}. (b) *Convergence of the solution* u *as* $\sigma \to 0^+$. k *and the accuracy are fixed at 8 and* 10^{-6}, *respectively.* (c) *Convergence of the solution as the wavelet order* k *increases.* σ *and the accuracy are fixed at* 0.1 *and* 10^{-8}, *respectively.* (d) *Convergence of the solution as the wavelet order increases while the accuracy tightens. If the wavelet order is* k, *the accuracy is* $10^{-(k-2)}$. σ *is fixed at* 0.1.

- Lines 69–81 wrap the fixed point iteration in a simple equation solver already available within MADNESS [27]. Line 69 calculates an initial guess for u, which is $G * \rho_{\text{vol}}$ (the potential of the charge in a uniform medium). Line 72 produces the surface charge ρ_{surf} and line 73 calculates the residual $u - G * (\rho_{\text{vol}} + \rho_{\text{surf}})$, respectively. Line 74 then computes the next iteration in u, and line 75 calculates the 2-norm of the update, $\|u_{j+1} - u_j\|_2$, where j is the iteration number. For efficiency of computation and memory, small coefficients are truncated after operations (notably the integral convolution operator) that can produce negligibly small boxes (e.g., due to conservative computation or due to smoothing properties of kernels). This is the purpose of the various calls to `truncate()`.
- Finally, lines 83–84 tabulate the potential at 501 points along a line from $(0, 0, 0)$ to $(5, 0, 0)$ for plotting.

Figure 2 displays numerical results obtained using this code that demonstrate convergence with respect to the various numerical parameters. In Figure 2(a) we plot the analytic solution to the problem, (3.4). Panel (b) shows the difference between the exact solution and the diffuse boundary approximations with $\sigma = 0.1$, 0.5, and 0.25 and fixed accuracy 10^{-6} (wavelet order $k = 8$). Panel (c) presents the difference between the exact solution and the diffuse boundary approximation with $\sigma = 0.1$ with an accuracy of 10^{-8} and $k = 6$, 8, or 10. Finally, panel (d) shows the difference between the exact solution and the diffuse boundary approximation with $\sigma = 0.1$, where we compute with accuracies 10^{-4} ($k = 6$), 10^{-6} ($k = 8$) and 10^{-8} ($k = 10$).

As more accuracy is requested, it is apparent that we obtain a similar increase in the accuracy of the result, which is observed whether or not we simultaneously increase the order of the basis. (The relationship between accuracy and k is a heuristic that attempts to maintain efficiency). The long-range asymptotic form of the potential is

$\epsilon_{\text{ext}}|\vec{r}|^{-1}$ and independent of σ, so errors in u for $|\vec{r}| \gg R$ are attributable entirely to the accuracy of computation. On the other hand, the internal potential, u for $|\vec{r}| < R$, has an error that is linear in σ and this dependence is shown in panel (b).

3.3. Molecular electronic structure. One of the initial targets when developing MADNESS was quantum chemistry, where electronic structure problems for atoms and molecules are solved using the density functional theory (DFT) or the Hartree–Fock methods [28, 29, 57]. Either of these approaches requires solving a set of nonlinear partial differential equations, which, for time-independent systems, are typically cast into an eigenvalue problem,

$$\left(-\frac{1}{2}\nabla^2 + V(\vec{x})\right)\psi(\vec{x}) = E\psi(\vec{x}). \tag{3.8}$$

$V(\vec{x})$ is the potential energy for an electron at position $\vec{x}$ and is itself dependent on $\psi(\vec{x})$ (thus the nonlinearity). The eigenvalues are interpreted as energies of molecular states and the eigenfunctions as molecular orbitals. We are interested in the lowest energy states. Unfortunately, fixed-point iteration of this differential equation converges to the highest energy states; hence numerical schemes must employ preconditioners. In chemistry, the standard approach for solving (3.8) expands $\psi(\vec{x})$ in a predefined, fixed set of basis functions, thereby converting (3.8) into a generalized matrix equation. However, for high-accuracy, the necessary basis sets are large and the resulting matrices dense, resulting in a computational cost that scales cubically with the system size.

Our MADNESS implementation follows a different approach [38] to utilize the computational strengths of MADNESS and to avoid the numerical problems associated with derivative operators on deeply refined regions. Specifically, the differential form of equations such as (3.8) suffer not just from the familiar need for powerful preconditioners to converge to the desired lowest eigenvalue, but potentially from severe loss of accuracy due to the unbounded spectrum of the differential operator. For example, it is not uncommon in our applications to dyadically refine 30 or more levels, and since $2^{30} \approx 10^9$, a single application of the Laplacian can amplify high-frequency noise by a factor of $O(10^{18})$. This can be catastrophic.

Thus, we recast (3.8) as an integral equation,

$$\psi(\vec{x}) = -2\int \mathrm{d}\vec{x}(-\nabla^2 - 2E)^{-1}V(\vec{x})\psi(\vec{x}), \tag{3.9}$$

which can be solved using a fixed point iteration without preconditioning to obtain the lowest eigenvalue(s). In practice, the iteration is wrapped in a nonlinear equation solver to ensure efficient convergence (much like the example in section 3.2). By solving for a set of spatially localized, orthonormal functions that span the space of the sought eigenfunctions (which are often quite delocalized), the effective computational cost is significantly reduced.

The Green's function for the bound-state ($E < 0$) Helmholtz equation (i.e., the integral kernel to the inverse of the pseudo-differential operator in (3.9)) that assumes free-space boundary conditions in three dimensions is

$$G(\vec{x}, \vec{x}') = \frac{1}{4\pi}\frac{e^{-\sqrt{-2E}}}{|\vec{x} - \vec{x}'|}. \tag{3.10}$$

This integral kernel not only incorporates the correct long-range asymptotic form for bound states molecular orbitals, but it can be efficiently applied using the separated

representations discussed in section 2.3. Capabilities of the molecular electronic structure code include energies for a wide range of functionals (including hybrids) and for many-body methods (second-order Møller–Plesset in six dimensions [14, 15] and configuration interaction [39]), localized orbitals for reduced scaling, forces [56], solvation [25], and linear-response for excited states and dynamic polarizabilities [49, 58, 55].

3.4. Nuclear physics. Nuclear physics is another successful application for MADNESS [24, 41, 42, 43]. Nuclear DFT extends the molecular DFT above to describe the complex superfluid many-fermion and boson system. A critical difference of NDFT is the use of two-component and four-component complex orbitals with particle spins and the appearance of resonance and continuum states. For example, the two-component complex wave-functions, u and v, and an associated pairing potential, Δ, extend the one-component orbital (ψ) in the molecular DFT to NDFT. These modifications ultimately result in the Hartree–Fock–Bogoliubov (HFB) equations,

$$\begin{bmatrix} h_\uparrow - \lambda_\uparrow & \Delta \\ \Delta^* & h_\downarrow - \lambda_\downarrow \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = E_i \begin{bmatrix} u_i \\ v_i \end{bmatrix}, \tag{3.11}$$

where $h_{\uparrow\downarrow}$ is a single particle Hamiltonian for the given component and $\lambda_{\uparrow\downarrow}$ is the chemical potential for that component. For the ultracold fermionic gas in the BEC-BCS (Bose–Einstein condensate—Bardeen–Cooper–Schriffer) crossover simulation, the oscillatory Fulde–Ferrell–Larkin–Ovchinnikov phase was found to have oscillating pairing gaps such that

$$h_{\uparrow\downarrow} = -\frac{1}{2}\nabla \cdot [\nabla \alpha_{\uparrow\downarrow}(r)] + V_{\uparrow\downarrow}(r). \tag{3.12}$$

$\alpha_{\uparrow\downarrow}$ is the kinetic energy density. $V_{\uparrow\downarrow}$ is the trapping potential, and the $\uparrow$ and $\downarrow$ symbols refer to spin. The HFB equations are solved with a complex version of the scheme used by the molecular density functional application with additional geometric constraints; that is, (3.11) is solved both by spectral approximation and then by an eigensystem calculation, followed by a corresponding Lippmann–Schwinger scattering equation.

3.5. Atomic and molecular physics. The above applications are all time-independent or introduce time-dependence only through response theory. More recently [54] we were interested in the effects of nonperturbative, few-cycle laser pulses on atoms and molecules, a problem that cannot be handled using the above ideas. Accordingly, we developed numerically robust and accurate techniques for evolving quantum wavepackets (and more general systems of partial differential or integral equations) forward in time. The familiar challenges of time evolution are exacerbated by adaptive refinement and truncation of the multiwavelet basis. Basis truncation introduces high-frequency noise, as do the inevitable discontinuities between adjacent subdomains. This noise must be removed from the simulation for accuracy and efficiency. Unfortunately, we cannot casually accept that frequencies beyond some band limit (cut-off) will be propagated inaccurately; for example, the wave function of the time-dependent Schrödinger equation (TDSE) depends upon accurate phase information. Our solution was to choose a band limit, propagate exactly below the band limit, and filter frequencies above it, as if we were essentially computing on a globally very fine, uniform mesh but still retaining the advantages of adaptive refinement.

The TDSE was solved [54] for hydrogenic atoms and other systems subjected to a strong, attosecond laser field, and the photoionization cross section and other observables were computed. The electron behavior was modeled by a three-dimensional

wave function, $\Psi(\vec{x},t)$, which, within the dipole approximation and in atomic units, evolves according to the TDSE,

$$i\frac{d\Psi(\vec{x},t)}{dt} = \left(-\frac{1}{2}\nabla^2 + V(\vec{x}) + \vec{E}(t)\cdot\vec{x}\right)\Psi(\vec{x},t), \tag{3.13}$$

where $\vec{E}$ is the electric field and V the atomic potential. The initial state, $\Psi(\vec{x},0)$, is the ground state, that is, the solution of (3.8) with the smallest E.

The most efficient evolution was performed with a fourth-order accurate, gradient-corrected exponential propagator [20] that is only slightly more expensive than the second-order accurate Trotter approximation. Crucial for efficient and accurate application was the realization that, while the kernel of the free-space propagator,

$$(2\pi it)^{-1/2}\exp\left(-\frac{|\vec{x}|^2}{2it}\right),$$

is infinite in extent and highly oscillatory, its combination with a projector onto a finite bandlimit is compact and also bandlimited.

4. Computational framework: The MADNESS parallel runtime. The MADNESS parallel runtime, like the numerical libraries of MADNESS, is an intuitive interface that allows the user to compose algorithms as a dynamically scheduled set of tasks that operate on objects in global namespaces. This high-level approach to parallel programming offers greater composability than that of the traditional message passing interface (MPI) and explicit thread programming (POSIX or C++ threads). The key runtime features include the use of

1. *global namespaces* for building applications that are composed of (global) objects interacting via remote methods;
2. *tasks* as first-class entities for fine-grained work decomposition;
3. *futures* [8] for expressing dependencies between the tasks.

These features permit the programmer to deal less with the explicit low-level details of computation and data partitioning (e.g., message passing between processes with disjoint data or threads operating on shared data) and focus more on high-level concepts central to the scientific domain of interest. An algorithm properly composed within the MADNESS runtime will (partially) hide costs of data communication and be tolerant to load imbalance due to dynamic, data-driven work scheduling.

All of these individual concepts appear in other packages, for example, in Cilk [16, 17], Charm++ [35, 36, 37], Intel Threading Building Blocks (TBB) [3], and other projects including ACE [47, 48] and PGAS languages [59]. Some of these features are also starting to make their way into mainstream programming languages. C++, for example, has included task-based concurrency with asynchronous function calls and futures since 2011. What makes the MADNESS runtime stand out is how the aforementioned features are composed to allow powerful construction of parallel programs. For example, MADNESS futures, unlike C++ and other future implementations, can refer not only to results of local tasks but also to those of *remote* tasks, thus directly supporting composition of *distributed* task-parallel algorithms.

For the sake of portability the MADNESS parallel runtime is implemented consistent with recent standards of C++ (2011) and MPI (3.0). Intranode parallelism is achieved through the Intel TBB library or, if TBB is not available on a particular platform, a similar thread runtime written in POSIX threads, Linux system calls, and native atomic operations. As of the writing of this manuscript, MADNESS is able

to use TBB on Intel, AMD, and IBM platforms. In the most common scenario, each node executes a single MPI process composed of the main application thread. The MADNESS runtime spawns a remote method invocation (RMI) thread and a pool of computational threads managed by a task queue; these threads are not directly accessible to the user. The main application thread runs sequentially through the main program and submits tasks (that can themselves generate more tasks) to the local or remote task queue.[1] The output of each task is encapsulated in a future that can serve as input to other tasks. Once its input futures are available (assigned), the task queue schedules the tasks for execution. Because MADNESS futures live in a global namespace, assigning a future can involve data movement to a remote process; this process is facilitated by the RMI thread in the receiving process.

By supporting the straightforward composition of a distributed task-based application, the MADNESS runtime allows programmers to focus on exposing concurrency by decomposing data and computation or by optimizing load balance. The low-level details of thread scheduling and synchronization, data movement, and communication hiding are automated. However, the MADNESS runtime provides access to enough low-level details of the architecture (e.g., process rank) that the placement of data and computation can be directly controlled by the user. This allows tight orchestration of algorithms with complex data flows, such as in the systolic loops used for computing localized electronic states in molecular electronic structure. These abilities, as provided by the MADNESS runtime, are also used by `TiledArray` [19, 18] (a framework for block-sparse tensor computations) to hide communication costs and withstand load imbalances in handling block-sparse data.

In addition to the above core capabilities, the MADNESS parallel runtime also provides higher-level constructs built on these features. One such feature is a distributed-memory, associative container used to store the functions and operators in the MADNESS numerical library. This container automates the placement and computation of distributed data. Other features include parallel algorithms (e.g., parallel for-each) and task-based collective operations. Many of these components interoperate via futures and, therefore, fit naturally within the task-based framework. For example, the task-based all-reduce algorithm, which is analogous to the `MPI_Allreduce` function, uses futures as both the input and output of the function, allowing a seamless connection between local computation tasks, collective communication, and remote-computation tasks.

5. Summary. Since the introduction and initial successful science application of MADNESS, it received an R&D 100 Award in 2011 [5] and has motivated at least three independent (and subsequent) implementations of the associated numerical methods in Japan [50], Norway [26], and the United States [52]. MADNESS itself is now thriving as an open-source, community project with financial support from multiple sources. Central to people's interest is the emphasis on a high-level of composition while maintaining guarantees of accuracy and speed. It is not that MADNESS is necessarily the fastest code for any particular problem, but rather that a user working on tractable problems will get the right answer with modest effort and without having to place unnecessary emphasis on computational details. However, the caveat of "tractability" is nontrivial—user attention is still needed to regularize some singularities, and the fast algorithms within MADNESS (notably the integral convolution kernels) do not yet include scattering operators, which is a subject of current research.

[1]The main thread can also act as a computational thread by executing tasks in the task queue while waiting for an event (e.g., while waiting inside a barrier).

Similarly, efficient computation is presently limited to simple domains and, as shown in [44], cumbersome techniques are presently necessary to accommodate even simple interior boundary conditions. Nevertheless, the numerical techniques have demonstrated their worth in a broad range of interesting applications in chemistry, physics, and elsewhere. In a similar fashion, the MADNESS parallel runtime is being successfully used for petascale computations independent of the numerical layer [18, 19], illustrating the power and utility of the massively threaded, task-based approach to computation.

For reference, commit `1b0843f` (dated December 11, 2015) in the MADNESS repository [1] was used for the code examples and discussion throughout this work.

Appendix A. Asymptotic decay of matrix elements. We briefly demonstrate the asymptotic rate of decay of the interaction between wavelets (or between wavelets and scaling functions), which is central to the effective sparsity of the matrix representation of operators in the wavelet basis. For the electrostatic potential, this results from a standard multipole expansion [32]. However, a slightly more general discussion permits us to comment on the types of operators for which the multiresolution approach is efficient without introducing additional techniques.

Let $f(\vec{x})$ and $g(\vec{y})$ be functions with disjoint, compact support such that concave surfaces enclosing the support of each function do not intersect, and let $T(|\vec{x}-\vec{y}|)$ be the integral kernel of interest. We assume T is infinitely differentiable away from the origin, where it may be singular. (Hypersingular operators can also be accommodated but require additional consideration.) The matrix element we seek is

$$\int \mathrm{d}^3\vec{x} \int \mathrm{d}^3\vec{y}\, f(\vec{x}) T(|\vec{x}-\vec{y}|) g(\vec{y}). \tag{A.1}$$

We now perform a Taylor series expansion of the operator in both $\vec{x}$ and $\vec{y}$ about $\vec{X}$ and $\vec{Y}$, respectively, with the latter chosen as points within the support of f and g, respectively. Consider the coefficient of a representative term $(x_p - X_p)^m (y_q - Y_q)^n$,

$$\begin{aligned} &\frac{1}{n!m!} \left(\int \mathrm{d}^3\vec{x}\, f(\vec{x}) (x_p - X_p)^m \right) \left(\int \mathrm{d}^3\vec{y}\, g(\vec{y}) (y_q - Y_q)^n \right) \\ &\qquad \times \left(\frac{\partial^{m+n}}{\partial x_p^m \partial y_q^n} T(|\vec{x}-\vec{y}|) \Big|_{\vec{x}=\vec{X},\vec{y}=\vec{Y}} \right), \end{aligned} \tag{A.2}$$

where the p and q subscripts denote components of the vectors.

Inside the first two sets of parentheses are the multipole moments of the functions, and the last set contains a derivative of the kernel. If the kernel is $|\vec{x}-\vec{y}|^{-1}$, then all of its $(m+n)$th-order derivatives are readily seen to decay as $\mathcal{O}(|\vec{X}-\vec{Y}|^{-(m+n+1)})$. Furthermore, when both f and g are wavelets (corresponding to the A block of the nonstandard form in (2.5)), both moments in (A.2) can only be nonzero if $m \geq k$ and $n \geq k$ (because at least the first k moments of f and g vanish). Thus, the matrix elements decay as $\mathcal{O}(|\vec{X}-\vec{Y}|^{-(2k+1)})$ or faster, noting that Alpert [6] chose the higher-index wavelets to have more vanishing moments. Similarly, if one of f and g is a wavelet and the other is a scaling function (corresponding to the B and C blocks of the nonstandard form) only one of m or n needs to be $\geq k$. Hence, those matrix elements decay as $\mathcal{O}(|\vec{X}-\vec{Y}|^{-(k+1)})$ or faster.

Thus, the sparsity of such a matrix representation arises from the smoothness of the kernel increasing with distance from the origin. Kernels such as

$$\frac{\exp(-a|\vec{X}-\vec{Y}|)}{|\vec{X}-\vec{Y}|}$$

for real $a > 0$ also have this property. However, if a is imaginary, the kernel oscillates and does not appear smooth until resolved at length scales shorter than the wavelength of the oscillation. In this case, the matrix representation (in the wavelet basis) of such an operator would be dense at coarse length scales.

REFERENCES

[1] MADNESS Code Repository, https://github.com/m-a-d-n-e-s-s/madness.
[2] OpenDX, http://www.opendx.org.
[3] Threading Building Blocks, https://www.threadingbuildingblocks.org.
[4] VisIt, https://wci.llnl.gov/simulation/computer-codes/visit.
[5] *Free framework for scientific simulation*, R&D Magazine, 53 (2011), p. 28.
[6] B. Alpert, *A class of bases in L^2 for the sparse representation of integral operators*, SIAM J. Math. Anal., 24 (1993), pp. 246–262.
[7] B. Alpert, G. Beylkin, D. Gines, and L. Vozovoi, *Adaptive solution of partial differential equations in multiwavelet bases*, J. Comput. Phys., 182 (2002), pp. 149–190.
[8] H. G. Baker, Jr., and C. Hewitt, *The incremental garbage collection of processes*, in Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages, ACM, New York, 1977, pp. 55–59.
[9] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual Revision* 3.5, Technical report ANL-95/11 Rev. 3.5, Argonne National Laboratory, 2014.
[10] G. Beylkin, V. Cheruvu, and F. Pérez, *Fast adaptive algorithms in the non-standard form for multidimensional problems*, Appl. Comput. Harmon. Anal., 24 (2008), pp. 354–377.
[11] G. Beylkin, R. Coifman, and V. Rokhlin, *Fast wavelet transforms and numerical algorithms* I, Comm. Pure Appl. Math., 44 (1991), pp. 141–183.
[12] G. Beylkin, R. Cramer, G. Fann, and R. J. Harrison, *Multiresolution separated representations of singular and weakly singular operators*, Appl. Comput. Harmon. Anal., 23 (2007), pp. 235–253.
[13] G. Beylkin and L. Monzón, *Approximation by exponential sums revisited*, Appl. Comput. Harmon. Anal., 28 (2010), pp. 131–149.
[14] F. A. Bischoff, R. J. Harrison, and E. F. Valeev, *Computing many-body wave functions with guaranteed precision: The first-order Møller–Plesset wave function for the ground state of helium atom*, J. Chem. Phys., 137 (2012), 104103.
[15] F. A. Bischoff and E. F. Valeev, *Computing molecular correlation energies with guaranteed precision*, J. Chem. Phys., 139 (2013), 114106.
[16] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, *Cilk: An efficient multithreaded runtime system*, in Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Santa Barbara, CA, 1995, pp. 207–216.
[17] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. L. Zhou, *Cilk: An efficient multithreaded runtime system*, J. Parallel Distrib. Comput., 37 (1996), pp. 55–69.
[18] J. A. Calvin, C. A. Lewis, and E. F. Valeev, *Scalable task-based algorithm for multiplication of block-rank-sparse matrices*, in Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms, IA3 '15, New York, 2015, ACM, pp. 4:1–4:8.
[19] J. A. Calvin and E. F. Valeev, *TiledArray: A massively-parallel, block-sparse tensor library written in C++*, https://github.com/valeevgroup/tiledarray/, 2015.
[20] S. A. Chin and C. R. Chen, *Fourth order gradient symplectic integrator methods for solving the time-dependent schrödinger equation*, J. Chem. Phys., 114 (2001), pp. 7338–7341.
[21] C. Cohen-Tannoudji, B. Diu, and F. Laloë, *Quantum Mechanics*, John-Wiley & Sons, West Sussex, UK, 2000.
[22] T. A. Driscoll, N. Hale, and L. N. Trefethen, eds., *Chebfun Guide*, Pafnuty Publications, Oxford, 2014.

[23] G. Fann, G. Beylkin, R. J. Harrison, and K. E. Jordan, *Singular operators in multiwavelet bases*, IBM J. Res. Devel., 48 (2004), pp. 161–171.
[24] G. I. Fann, J. Pei, R. J. Harrison, J. Jia, J. Hill, M. Ou, W. Nazarewicz, W. A. Shelton, and N. Schunck, *Fast multiresolution methods for density functional theory in nuclear physics*, J. Phys. Conf. Ser., 180 (2009), 012080.
[25] J. Fosso-Tande and R. J. Harrison, *Implicit solvation models in a multiresolution multiwavelet basis*, Chem. Phys. Lett., 561–562 (2013), pp. 179–184.
[26] L. Frediani, E. Fossgaard, T. Flå, and K. Ruud, *Fully adaptive algorithms for multivariate integral equations using the non-standard form and multiwavelets with applications to the Poisson and bound-state Helmholtz kernels in three dimensions*, Mol. Phys., 111 (2013), pp. 1143–1160.
[27] R. J. Harrison, *Krylov subspace accelerated inexact Newton method for linear and nonlinear equations*, J. Comput. Chem., 25 (2004), pp. 328–334.
[28] R. J. Harrison, G. I. Fann, T. Yanai, and G. Beylkin, *Multiresolution quantum chemistry in multiwavelet bases*, Lecture Notes in Comput. Sci. 2660, Springer, Heidelberg, 2003, pp. 103–110.
[29] R. J. Harrison, G. I. Fann, T. Yanai, Z. Gan, and G. Beylkin, *Multiresolution quantum chemistry: Basic theory and initial applications*, J. Chem. Phys., 121 (2004), pp. 11587–11598.
[30] A. Henderson, *ParaView Guide, A Parallel Visualization Application*, Kitware, 2007.
[31] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the trilinos project*, ACM Trans. Math. Software, 31 (2005), pp. 397–423.
[32] J. D. Jackson, *Classical Electrodynamics*, 3rd ed., John Wiley & Sons, West Sussex, UK, 1999.
[33] J. Jia, R. Harrison, and G. Fann, *Fast transform from an adaptive multi-wavelet representation to a partial Fourier representation*, J. Comput. Phys., 229 (2010), pp. 5870–5878.
[34] J. Jia, J. C. Hill, G. I. Fann, and R. J. Harrison, *Multiresolution fast methods for a periodic 3-D Navier–Stokes solver*, in Proceedings of DCABES 2009: The 8th International Symposium on Distributed Computing and Applications to Business, Engineering, and Science, 2009, pp. 13–16.
[35] L. V. Kalé, *Parallel Programming with CHARM: An Overview*, Technical report, Parallel Programming Laboratory, University of Illinois at Urbana-Champaign, 1993.
[36] L. V. Kalé and S. Krishnan, *CHARM++: A portable concurrent object oriented system based on C++*, in Proceedings of the 8th Annual Conference on Object-oriented Programming Systems, Languages, and Applications, Anrdeas Paepcke, ed., ACM, New York, 1993, pp. 91–108.
[37] L. V. Kalé, B. Ramkumar, A. B. Sinha, and V. A. Saletore, *The CHARM Parallel Programming Language and System: Part* II*—The Runtime System*, Technical report, Parallel Programming Laboratory, University of Illinois at Urbana-Champaign, 1994.
[38] M. H. Kalos, *Monte Carlo calculations of the ground state of three- and four-body nuclei*, Phys. Rev., 128 (1962), pp. 1791–1795.
[39] J. S. Kottmann, S. Höfener, and F. A. Bischoff, *Numerically accurate linear response-properties in the configuration-interaction singles (CIS) approximation*, Phys. Chem. Chem. Phys., 17 (2015), doi:10.1039/C5CP00345H.
[40] A. Logg, K.-A. Mardal, and G. N. Wells, eds., *Automated Solution of Differential Equations by the Finite Element Method*, Springer, Heidelberg, 2012.
[41] J. C. Pei, G. I. Fann, R. J. Harrison, W. Nazarewicz, J. Hill, D. Galindo, and J. Jia, *Coordinate-space Hartree–Fock–Bogoliubov for superfluid Fermi systems in large boxes*, J. Phys. Conf. Ser., 402 (2012), 012035.
[42] J. C. Pei, G. I. Fann, R. J. Harrison, W. Nazarewicz, Y. Shi, and S. Thornton, *Adaptive multi-resolution 3D Hartree–Fock–Bogoliubov solver for nuclear structure*, Phys. Rev. C, 90 (2014), 024317.
[43] J. C. Pei, M. V. Stoitsov, G. I. Fann, W. Nazarewicz, N. Schunck, and F. R. Xu, *Deformed coordinate-space Hartree–Fock–Bogoliubov approach to weakly bound nuclei and large deformations*, Phy. Rev. C, 78 (2008), 064306.
[44] M. G. Reuter, J. C. Hill, and R. J. Harrison, *Solving PDEs in irregular geometries with multiresolution methods* I: *Embedded Dirichlet boundary conditions*, Comput. Phys. Commun., 183 (2012), pp. 1–7.
[45] M. G. Reuter, M. A. Ratner, and T. Seideman, *Laser alignment as a route to ultrafast control of electron transport*, Phys. Rev. A, 86 (2012), 013426.
[46] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[47] D. C. Schmidt, *The ADAPTIVE Communication Environment (ACE)*.
[48] D. C. Schmidt and S. D. Huston, *C++ Network Programming: Systematic Reuse with ACE and Frameworks*, Vol. 2, Pearson Education, Cranbury, NJ, 2002.
[49] H. Sekino, Y. Maeda, T. Yanai, and R. J. Harrison, *Basis set limit Hartree–Fock and density functional theory response property evaluation by multiresolution multiwavelet basis*, J. Chem. Phys., 129 (2008), 034111.
[50] H. Sekino, T. Okamoto, and S. Hamada, *Solution of wave-equation using multiresolution multiwavelet basis function*, in Proceedings of the 2010 International Conference on Wavelet Analysis & Pattern Recognition (ICWAPR), 2010, pp. 355–358.
[51] B. D. Sullivan, D. Weerapurage, and C. Groer, *Parallel algorithms for graph optimization using tree decompositions*, in Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013, pp. 1838–1847.
[52] E. F. Valeev and F. A. Bischoff, *private communication*, 2011.
[53] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.
[54] N. Vence, R. Harrison, and P. Krstić, *Attosecond electron dynamics: A multiresolution approach*, Phys. Rev. A, 85 (2012), 033403.
[55] T. Yanai, G. I. Fann, G. Beylkin, and R. J. Harrison, *Multiresolution quantum chemistry in multiwavelet bases: Excited states from time-dependent Hartree–Fock and density functional theory via linear response*, Phys. Chem. Chem. Phys., 17 (2015), pp. 31405–31416.
[56] T. Yanai, G. I. Fann, Z. Gan, R. J. Harrison, and G. Beylkin, *Multiresolution quantum chemistry in multiwavelet bases: Analytic derivatives for Hartree–Fock and density functional theory*, J. Chem. Phys., 121 (2004), pp. 2866–2876.
[57] T. Yanai, G. I. Fann, Z. Gan, R. J. Harrison, and G. Beylkin, *Multiresolution quantum chemistry in multiwavelet bases: Hartree–Fock exchange*, J. Chem. Phys., 121 (2004), pp. 6680–6688.
[58] T. Yanai, R. J. Harrison, and N. C. Handy, *Multiresolution quantum chemistry in multiwavelet bases: Time-dependent density functional theory with asymptotically corrected potentials in local density and generalized gradient approximations*, Mol. Phys, 103 (2005), pp. 413–424.
[59] K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. L Graham, P. Hargrove, P. Hilfinger, P. Husbands, et al., *Productivity and performance using partitioned global address space languages*, in Proceedings of the 2007 International Workshop on Parallel Symbolic Computation, ACM, New York, 2007, pp. 24–32.