Parallel Implementation of Time-Dependent Density Functional Theory *

W. Russell Burdick [†] Yousef Saad [†] Leeor Kronik [‡] Igor Vasiliev [§]
Manish Jain [¶] James R. Chelikowsky [¶]

July 7, 2003

Abstract

We present a massively parallel implementation of time-dependent density functional theory in real space, aimed at computing optical absorption spectra of realistic systems with hundreds of atoms from first principles. We provide details of the formalism and discuss its implementation, optimization, and efficient parallelization, as well as remaining limitations, in detail. The capabilities of the code are illustrated by calculations of optical properties of hydrogenated silicon quantum dots.

1 Introduction

A central goal of modern computational materials science is the *ab initio* prediction of various materials properties, using only information about the constituent species and the laws of quantum physics [1]. Realization of this goal would allow scientists to explain and predict surprising material phenomena found experimentally, and ultimately to predict the existence of new materials and determine their properties. One of the most popular approaches for such computations is to use *ab initio* pseudopotentials within density functional theory (DFT) [2]. This approach has been used successfully for predicting mechanical, chemical, and electronic properties of many classes of solids, liquids, molecules, and more.

The pseudopotential-density functional approach weds two physically reasonable approximations that result in a dramatic reduction of computational complexity. Within density functional theory, the original N-electron problem is mapped into an effective one-electron problem, where all non-classical electron interactions (namely, exchange and correlation) are subsumed into an additive one-electron potential that is a functional of the charge density. While this mapping is formally exact, it is approximate in practice because the exact functional is unknown. The most common approximate functional, which is used in this paper, is the local density approximation (LDA), where the exchange-correlation functional is taken to be a local function of the charge density. Mathematically, DFT converts a differential eigenvalue problem in 3N spatial coordinates (which is practically intractable for all but the smallest systems) into a standard eigenvalue problem with only 3 spatial coordinates.

*Work supported by NSF grant NSF/ITR-0082094 and by the Minnesota Supercomputing Institute
[†]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455
[‡]Dept. of Materials and Interfaces, Weizmann Institute of Science, Rehovoth 76100, Israel
[§]Dept. of Physics, New Mexico State University, Las Cruces, NM 88003
[¶]Department of Chemical Engineering and Materials Science, University of Minnesota, Minneapolis, MN 55455

In the pseudopotential approximation, only valence electrons (the ones forming chemical bonds) are treated explicitly. Core electrons are suppressed by replacing the true atomic potential with an effective "pseudopotential" that takes the effect of core electrons into account. This facilitates DFT calculations in two ways: First, the pseudopotential is a smooth, slowly-varying potential, whereas the true atomic potential is rapidly varying and has a $\sim 1/r$ singularity at the nuclear position. This makes the attainment of a converged numerical solution significantly easier. Second, treating only valence electrons reduces the number of eigenvalues that need to be found, often by as much as an order of magnitude.

One of the most significant limitations of the above approach is that it is unsuitable for the computation of electronic excitations in general, and optical properties in particular. This is because traditional, time-independent DFT is inherently a ground-state theory [2]. A generalized, *time-dependent* DFT (TDDFT) formalism that allows for computations of excited state properties has been developed and applied successfully to a wide range of atoms and small molecules [3, 4]. Optical properties of larger systems of molecules, clusters, or "quantum dots" (small fragments of bulk material) in the range of many hundreds of atoms are much harder to address despite their outstanding importance in physics. This is primarily due to severe computational limitations.

In this article, we present a massively parallel implementation of the pseudopotential-TDDFT formalism that easily allows for computation of optical properties of systems with many hundreds of atoms. This was made possible by using a real-space implementation of a frequency-domain formulation of the TDDFT equations and by judicious analysis of computational bottlenecks. The article is arranged as follows. First, we recapitulate the frequency-domain formulation of TDDFT and explain its advantages. Next, we present our real-space implementation of this frequency domain formulation and discuss the merits of this implementation. We then proceed with a detailed discussion of various elements of the algorithm and present strategies for parallelization and optimization based on both mathematical and computer science considerations. Finally, we present calculations for hydrogenated silicon quantum dots performed using the present approach and use them to demonstrate timing and scaling issues.

2 Formalism

The frequency-domain formulation of TDDFT implemented in our code is the one presented by Casida [4]. For physical insight into this formulation, we refer the reader to the original publication. Here we merely present the pertinent equations without proof so that we can analyze their computational implications in the next sections. For simplicity, we shall restrict ourselves to the spin-independent form of the equations.

A major advantage of the frequency-domain formulation is that the need for any explicit solution of a time-dependent equation is eliminated. Instead, we first solve the usual, time *independent* formulation of the pseudopotential-DFT method [2]:

$$\left(-\frac{\nabla^2}{2} + \sum_{\vec{R}_a} V_{ps}(\vec{r} - \vec{R}_a) + V_H[\rho(\vec{r})] + V_{xc}[\rho(\vec{r})]\right)\psi_n(\vec{r}) = E_n\psi_n(\vec{r}). \tag{1}$$

Atomic units are used throughout. In Eq. (1), E_n and $\psi_n(\vec{r})$ are the n^{th} single-electron eigenvalue and eigenfunction, respectively; $\rho(\vec{r})$ is the valence charge density, given by $\rho(\vec{r}) = -\sum_n f_n|\psi_n(\vec{r})|^2$, where f_n are the electron-occupation numbers associated with each eigenfunction, as determined from Pauli's exclusion principle; $V_{ps}(\vec{r}-\vec{R}_a)$ is the pseudopotential associated with an atom situated

at $\vec{R}_a$, $V_H[\rho(\vec{r})]$ is the potential due to electron-electron interaction, and $V_{xc}[\rho(\vec{r})]$ is the above-mentioned exchange-correlation potential.

Just like DFT, TDDFT is also exact in principle but approximate in practice. In addition to an assumption about the *spatial* nature of the exchange-correlation functional made in any practical DFT calculation, one must now make a further assumption about its *temporal* form. In the adiabatic approximation, which is the one used most commonly in actual computations, one simply assumes that at any given time, the exchange-correlation potential does not depend on the history of the charge density, only on its current value. Under this assumption, the eigenvalues, eigenfunctions and occupation numbers found by solving Eq. (1) are used to construct a coupling matrix, $K_{ij,kl}$, in the form:

$$K_{ij,kl} = 2\iint \psi_i^*(\vec{r})\psi_j(\vec{r})\left(\frac{1}{|\vec{r}-\vec{r'}|} + \frac{\delta V_{xc}[\rho(\vec{r})]}{\delta\rho(\vec{r'})}\right)\psi_k(\vec{r'})\psi_l^*(\vec{r'})d\vec{r}d\vec{r'}, \tag{2}$$

where $\delta/\delta\rho(\vec{r'})$ denotes a functional derivative.

Under the local density approximation, the exchange-correlation functional is spatially local and the functional derivative reduces to $\delta V_{xc}[\rho(\vec{r})]/\delta\rho(\vec{r'}) = [dV_{xc}[\rho(\vec{r})]/d\rho(\vec{r})]\delta(\vec{r}-\vec{r'})$, where $\delta(\cdot)$ is the Dirac delta function. Eq. (2) can then be expressed as a sum of a double integral and a single integral:

$$K_{ij,kl} = 2\iint \psi_i^*(\vec{r})\psi_j(\vec{r})\frac{1}{|\vec{r}-\vec{r'}|}\psi_k(\vec{r'})\psi_l^*(\vec{r'})d\vec{r}d\vec{r'} + 2\int \psi_i^*(\vec{r})\psi_j(\vec{r})\frac{dV_{xc}[\rho(\vec{r})]}{d\rho(\vec{r})}\psi_k(\vec{r})\psi_l^*(\vec{r})d\vec{r}. \tag{3}$$

The coupling matrix, $K_{ij,kl}$, is then used as a kernel for forming a second eigenvalue problem - the TDDFT one - in the form:

$$QF_I = \Omega_I^2 F_I, \tag{4}$$

where

$$Q_{ij,kl} = \delta_{ik}\delta_{jl}\omega_{kl}^2 + 2\sqrt{\lambda_{ij}\omega_{ij}}K_{ij,kl}\sqrt{\lambda_{kl}\omega_{kl}}, \tag{5}$$

$\lambda_{ij} = f_i - f_j$ is an occupation number difference, and $\omega_{ij} = E_j - E_i$ is an eigenvalue difference. The eigenvalues, Ω_I, indicate optical-absorption transition energies. The strength of the absorption associated with the transition energy Ω_I, denoted by f_I, is related to the eigenvector F_I by:

$$f_I = \frac{2}{3}\sum_{\beta=1}^{3} |B_\beta^T\, R^{1/2}\, F_I|^2, \tag{6}$$

where

$$R_{ij,kl} = \delta_{ik}\delta_{jl}\lambda_{kl}\omega_{kl}; \qquad (B_\beta)_{ij} = \int \psi_i^*(\vec{r})r_\beta\psi_j(\vec{r}), \tag{7}$$

and $\{r_1, r_2, r_3\} = \{x, y, z\}$. Ω_I and f_I define the absorption spectrum uniquely and the TDDFT computation is complete.

3 Implementation

The above-described formalism indicates that three major computational blocks, shown schematically in Fig. 1, must be executed sequentially. The first part computes the DFT solution for the

eigenvalues, occupations, and wave functions. Next, this output is used as input for the computation of the coupling matrix (Eq. (3)). Finally, the coupling matrix serves as input for a computation of the eigenvalues and eigenvectors of it. Those are used to compute the optical absorption spectrum of the system.

Each of these three parts represents a distinct function and is implemented in a separate program. A complete computation consists of pipelining these programs sequentially. Massive parallelization is exploited extensively in each of the three programs. In this section, we provide a detailed description of the three programs.

3.1 The Real-Space Finite Difference Approach

According to the above formalism, the computation of the optical absorption must start with a solution of the conventional pseudopotential-DFT equation - Eq. (1). We chose to implement the pseudopotential-DFT code directly on a real-space grid using a finite difference approach. This is because the localization of the system (molecule, cluster, quantum dot,...) in space limits the number of grid points needed, and because parallelization schemes based on partitioning of real space into sub-regions are highly effective. We will not elaborate on the pseudopotential-DFT code because it has been discussed extensively elsewhere in terms of both its underlying physics [5, 6] and its computer science aspects [7, 8]. Here, we confine ourselves to a presentation of the real-space mapping and the finite difference approach, because the same mapping is used for the TDDFT computation.

Our real-space mapping is shown schematically in Fig. 2. Both potentials and wave functions are set up on a simple cartesian three-dimensional grid within a spherical domain. The grid points inside the sphere are described by their discrete space coordinates, (x, y, z). Outside the boundary domain, wave functions are required to vanish. The Laplacian term in Eq. (1) is approximated by a high-order finite-difference expansion, which replaces spatial derivatives with a weighted sum of the wave-function values at neighboring grid points:

$$\nabla^2\psi_i(x,y,z) = \frac{1}{h^2}\sum_{n=-N}^{N} C_{N,n}\left[\psi_i(x+nh,y,z) + \psi_i(x,y+nh,z) + \psi_i(x,y,z+nh)\right], \qquad (8)$$

where h is the grid spacing and $C_{N,n}$ are the coefficients in the order N finite difference expansion for the second derivative [9]. In the context of Eq. (8), two points are defined as neighbors if the vector connecting them is parallel to one of the three major axes and the points are up to N grid points away from each other. For a given accuracy of calculation, the finite-difference order should be chosen as a compromise between having a fine grid and a large but very sparse Hamiltonian matrix, and a coarse grid with a smaller but less sparse Hamiltonian. We find that a value of N ranging between 4 and 6 is typically a good choice [5].

3.2 Coupling Matrix Construction

Equation (3) consists of a sum of two integrals which can be computed separately. The first, double integral can be rewritten as:

$$\iint \psi_i^*(\vec{r})\psi_j(\vec{r})\frac{1}{|\vec{r}-\vec{r'}|}\psi_k(\vec{r'})\psi_l^*(\vec{r'})d\vec{r}d\vec{r'} = \int d\vec{r'}\psi_k(\vec{r'})\psi_l^*(\vec{r'})\int d\vec{r}\frac{1}{|\vec{r}-\vec{r'}|}\psi_i^*(\vec{r})\psi_j(\vec{r}) \qquad (9)$$

if we define

$$\rho_{ij}(\vec{r}) \equiv \psi_i^*(\vec{r})\psi_j(\vec{r}), \tag{10}$$

and

$$V_{ij}(\vec{r'}) \equiv \int d\vec{r} \frac{1}{|\vec{r} - \vec{r'}|} \psi_i^*(\vec{r})\psi_j(\vec{r}), \tag{11}$$

then we see that

$$V_{ij}(\vec{r'}) = \int \frac{\rho_{ij}(\vec{r})}{|\vec{r} - \vec{r'}|} d\vec{r}. \tag{12}$$

But with the appropriate boundary conditions, this is well-known to be the solution of the following Poisson equation:

$$\nabla^2 V_{ij}(\vec{r'}) = -4\pi\rho_{ij}(\vec{r'}). \tag{13}$$

Note that ρ_{ij} is not a physical charge density and V_{ij} is not a physical potential. However, they do have charge and potential units, respectively, and obey the same Poisson relation that a "true" charge density and Coulomb potential do.

While it is possible to use a direct summation to evaluate Eq. (12), other computational methods, such as the conjugate gradient method (with various enhancements) or the multigrid method, are far more efficient at solving in Eq. (13) [10].

Once the Poisson system is solved to find $V_{ij}(\vec{r'})$, we are left with the following form of Eq. (3):

$$K_{ij,kl} = \int d\vec{r'} \rho_{lk}(\vec{r'}) V_{ij}(\vec{r'}) + \int d\vec{r} \rho_{ij}(\vec{r}) \frac{dV_{xc}[\rho(\vec{r})]}{d\rho(\vec{r})} \rho_{lk}(\vec{r}) \tag{14}$$

The two single integrals in Eq. (14) must be computed by direct summation but this can be done for both of them at the same time:

$$K_{ij,kl} = \int d\vec{r} \Big[V_{ij}(\vec{r}) + \rho_{ij}(\vec{r}) \frac{dV_{xc}[\rho(\vec{r})]}{d\rho(\vec{r})} \Big] \rho_{lk}(\vec{r}). \tag{15}$$

In practice, we solve Eq. (13) in two phases. First, we set up the appropriate boundary conditions. Second, we solve for $V_{ij}(\vec{r'})$. Once the equation is solved, a third and final step in determining the coupling matrix $K_{ij,kl}$ is the evaluation of Eq. (15). Importantly, Eq. (13) needs to be solved only *once per row* of $K_{ij,kl}$, whereas Eq. (15) needs to be evaluated for each element. The three steps are shown in Fig. 3 and explained in detail in the following.

3.2.1 Boundary Condition Setup

Because we are dealing with a confined system, the appropriate boundary condition on the wave functions is that they must vanish outside the spherical domain. However, this null boundary condition is incorrect for potentials because potentials do not necessarily vanish even outside the region where the charges generating them are found. It then becomes necessary to set up the potential at the domain edge, in order to serve as the correct boundary condition for the potential distribution inside the domain.

To explain how this is implemented in practice, we define four types of grid points: "internal points", which are points within the sphere; "boundary points", which are points within the sphere that have neighbors outside the sphere; "boundary neighbors", which lie outside the sphere and have neighboring points inside the sphere; and "external points", which lie outside of the sphere.

Note that the term "neighbors" is used in the context of Eq. (8) above. Also note that a boundary neighbor is a special case of an external point and a boundary point is a special case of an internal point. A simplified, one dimensional example of this terminology is shown in Fig. 4. In this diagram, we show that if we are considering an order 3 expansion, then three of the four internal points shown are boundary points and three of the four external points shown are boundary neighbors.

Calculating the Coulomb potential, V, at any point with spherical coordinates (R, θ, ϕ) outside a sphere can be easily achieved using the standard expression for the Coulomb potential due to a distribution of effective point charges, e_a, situated at (r_a, θ_a, ϕ_a):

$$V(R,\theta,\phi) = \sum_a \frac{e_a}{|\vec{R} - \vec{r_a}|}. \tag{16}$$

Eq. (16) is physically transparent, but for all but the smallest systems it is impractical to use. As the system size increases, the large number of division operations required quickly renders the computation very expensive. It is therefore better to use an alternative approach, where $V(R, \theta, \phi)$ is expressed as an infinite sum over multipole terms, as shown in Eq. (17):

$$\begin{aligned} V(R,\theta,\phi) &= \sum_{\ell=0}^{\infty} \left\{ \frac{1}{R^{\ell+1}} \left[\sum_a \left[e_a r_a^\ell P_\ell^0(cos(\theta_a)) \right] P_\ell^0(cos(\theta)) \right. \right. \\ &+ 2 \cdot \mathrm{Re} \left\{ \sum_{m=1}^{\ell} \left[\frac{(\ell-m)!}{(\ell+m)!} \sum_a \left[e_a r_a^\ell P_\ell^m(cos(\theta_a)) e^{im\phi_a} \right] P_\ell^m(cos(\theta)) e^{-im\phi} \right] \right\} \left. \left. \right] \right\}. \end{aligned} \tag{17}$$

The derivation of this equation is given in Appendix A. In light of Eq. (13), $V_{ij}(R, \theta, \phi)$ is obtained by simply substituting ρ_{ij} for e_a in Eq. (17). For external points, this is an *exact* result and is strictly equivalent to the Coulomb form given above. Clearly it is not practical to evaluate the infinite sum in Eq. (17), but this is not required. Due to the factor of $1/R^{\ell+1}$ in Eq. (17) the series converges very rapidly and the summation can be truncated at a finite number, ℓ_{max}. We find that an ℓ_{max} value of 6 to 9 produces very good results with the computation remaining tractable.

The effective potential at the boundary neighbor points needs to be taken into account when expressing the Laplacian of the wave function at boundary points as a finite difference expansion. In practice, we compute a correction term for each boundary point, $c(r_a, \theta_a, \phi_a) = c(\vec{r})$, as:

$$c(\vec{r}) = \sum_{\text{boundary neighbors}} C_{N,n} V_{ij}(R,\theta,\phi), \tag{18}$$

where $C_{N,n}$ are the finite difference expansion coefficients of Eq. (8), determined according to the number of grid points between the boundary point and the boundary neighbor point. This correction is then used to modify the right hand side of Eq. (13):

$$\nabla^2 V_{ij}(\vec{r}) = -4\pi \rho_{ij}^{\text{corrected}}(\vec{r}) = -4\pi[\rho_{ij}(\vec{r}) - c(\vec{r})]. \tag{19}$$

We can then proceed with solving Eq. (19), which inherently satisfies the necessary boundary conditions.

An optimized implementation of Eq. (17) requires attention to several aspects involving the associated Legendre polynomials. First, one should avoid the temptation of calculating the formulas for the various associated Legendre polynomials recursively [11]. Instead, explicit formulas should be used. This is because the recursive definition involves many repeated calculations and will lead to a considerable slow-down. Second, powers of $x = cos(\theta)$ and $y = sin(\theta)$ that occur throughout the computation of the associated Legendre polynomials should be calculated only once, in the

most efficient manner possible. Third, the value of the associated Legendre polynomials at a given grid point is independent of the effective charge density at that point and is therefore the same for all elements of the coupling matrix. It is then better to store the value of those polynomials for each internal and boundary neighbor point, rather then re-evaluating them again for every row of the matrix. This requires an additional floating point array of size [number of relevant points]*[$\ell_{max}(\ell_{max}+3)/2$]. If this additional memory is prohibitive one could re-evaluate the polynomials anyway, at a performance cost.

It is also advisable to locally store the value of $V_{ij}(R,\theta,\phi)$ at each boundary neighbor inside the boundary condition subroutine. This is because the same boundary neighbor may be needed multiple times in the evaluation of Eq. (18), as it may neighbor multiple internal points (see Fig. 4). While this may seem like a small effect for the simple one dimensional example given in Fig. 4, in a practical three-dimensional case we could be considering as many as 9 neighbors in 6 different directions. Neglecting to store V_{ij} would then result in more than a five-fold increase in the number of multipole expansions computed.

In summary of this sub-section, the major steps of this computation are presented as pseudo-code in Algorithm 3.2.1.

Algorithm 3.2.1: Boundary Condition Calculation

Input: ρ_{ij}, ℓ_{max}, $ndim$, N.
Output: $\rho_{ij}^{\text{corrected}}$.

$q(\ell,m) = 0$
for $\ell = 0, \ell_{max}$ **do**
  for $m = 0, \ell$ **do**
    $c(\ell,m) = \frac{(\ell-m)!}{(\ell+m)!}$
  end for
end for
for each internal grid point, (r_a,θ_a,ϕ_a) **do**
  for $\ell = 0, \ell_{max}$ **do**
    for $m = 0, \ell$ **do**
      $q(\ell,m) = q(\ell,m) + \rho_{ij}(r_a,\theta_a,\phi_a) r_a^{\ell} P_\ell^m(cos(\theta_a)) e^{im\phi_a}$
    end for
  end for
end for
for each grid point, (R,θ,ϕ) **do**
  if this point is a boundary neighbor **then**
    $V_{ij}(R,\theta,\phi) = 0$
    for $\ell = 0, \ell_{max}$ **do**
      for $m = 0, \ell$ **do**
        $V_{ij}(R,\theta,\phi) = V_{ij}(R,\theta,\phi) + 2 \cdot \mathrm{Re}\{1/\mathrm{R}^{\ell+1}\mathrm{c}(\ell,\mathrm{m})\mathrm{q}(\ell,\mathrm{m})\mathrm{P}_\ell^{\mathrm{m}}(\cos(\theta))\mathrm{e}^{-\mathrm{im}\phi}\}$
      end for
    end for
  end if
end for
for each internal point, (r_a,θ_a,ϕ_a) **do**
  $\rho_{ij}^{\text{corrected}}(r_a,\theta_a,\phi_a) = \rho_{ij}(r_a,\theta_a,\phi_a)$
  for each of its neighbors, (R,θ,ϕ) **do**
    if (R,θ,ϕ) is a boundary neighbor **then**
      $\rho_{ij}^{\text{corrected}}(r_a,\theta_a,\phi_a) = \rho_{ij}^{\text{corrected}}(r_a,\theta_a,\phi_a) - C_{N,n} V_{ij}(R,\theta,\phi)$

29: **end if**
30: **end for**
31: **end for**

3.2.2 Solving the Poisson Equation

In a finite difference representation, Eq. (19) is transformed into an equivalent algebraic system $Ax = b$, where b is - $4\pi\rho_{ij}^{\text{corrected}}$, x is the potential, V_{ij}, and A is the matrix representing the Laplacian operator. The finite-difference representation of this operator (Eq. (8)) involves a maximum of $6N$ neighbors (less than that near the boundary), with N being the finite difference order. As mentioned above, in our application N is smaller than 10 and typically between 4 and 6. Consequently, a square matrix representation of the Laplacian operator would be very sparse. We opted to use a readily available conjugate gradient (CG) solver from the sparse matrix package SPARSKIT [12]. In its non-preconditioned form, this algorithm refers to the matrix only through matrix-by-vector products. In fact, the matrix is not explicitly required. Instead, we perform the matrix-by-vector products by using the finite difference "stencil" of the operator which acts directly on the input vector. This results in significant savings in memory usage.

Because a large number of vector operations may be required by the CG algorithm, efficient memory access becomes crucial to achieving good performance. Ideally, one would wish to minimize the address-space distance between subsequent memory accesses, known as the "stride length". Great care was exercised to structure our code to obtain unit stride length whenever possible.

A common approach to enhance the performance of the CG algorithm is the use of preconditioning. Preconditioning consists of replacing the original $Ax = b$ system with a system of the form $L^{-1}AL^{-T}(L^Tx) = L^{-1}b$, where LL^T is some approximate Cholesky factorization of A [13]. The goal is to reduce the number of CG iterations required for convergence. Among the simplest preconditioners available are techniques that perform a Gaussian elimination process (or a Cholesky factorization) and drop small terms that are filled-in during the elimination, according to various rules. (Note: A fill-in is defined as a new non-zero entry introduced during a factorization in a location of the matrix where there was initially a zero entry). The two most common strategies are the level-of-fill incomplete factorization, IC(k), and the threshold based factorization, ICT(k,τ)[14]. The level-of-fill approach associates a level for each fill-in introduced by a recursive rule that gives less weight to fill-ins introduced earlier. Then dropping is performed according to the weight. In the threshold based factorization, entries are dropped according to their numerical values, relative to the row norm or the diagonal elements. In ICT(k,τ), which is employed in our codes, the threshold parameter τ is used to drop small terms according to their size relative to the initial norm of the row. The parameter k is then used to keep the k largest elements in the resulting row.

This incomplete factorization is performed only once, with the decomposition applied at each CG iteration. Preconditioning usually results in a substantial reduction in the number of steps required for the algorithm to converge. There is, however, a trade-off associated with preconditioning. The higher the fill level, the fewer steps the CG algorithm will take to converge. On the other hand, higher fill factorizations require more memory and are more expensive to compute as well as to apply. In general, we found that taking τ close to 0.001 and k between 8 and 20 leads to a good balance between performance and memory requirements.

While the Incomplete Cholesky preconditioning yielded a sizable reduction in the number of steps (typically by a factor of 6 to 10), the resulting gain in time was much smaller (typically the computation was up to 50% faster). We note that better gains have been reported in the literature [15]. There are several reasons for this. First, the non-preconditioned CG iteration already

does fairly well, relatively speaking, requiring around 150 to 200 steps to converge. Second, the usual performance gains attributed to preconditioning in the literature are often related to two-dimensional problems. A two-dimensional Poisson problem is usually harder to solve using CG than a three-dimensional one because the eigenvalues are more closely spaced. Therefore, preconditioning is expected to benefit a two-dimensional problem more than a three-dimensional one. Third, the literature results typically refer to a standard 5-point centered difference discretization of the Laplacian. We are not aware of studies of the effects of using a higher order discretization. Finally, the forward-backward solutions required when applying the preconditioner are not easy to optimize because they lead to sequential and irregular computations. This difficulty can be partly alleviated by using a preconditioner based on a lower order discretization of the Laplacian operator, while maintaining the high order discretization for use in the matrix-vector products. Because the order could not be substantially lowered without affecting the preconditioning accuracy, this did not improve performance substantially.

Two additional directions for future improvements that we are exploring are the use of block CG methods [16] and the use of multi-grid techniques [17]. Block CG takes advantage of the fact that many different linear systems are to be solved independently with the same matrix A. In our case, Eq. (19) is solved many times with the same matrix, specifically once for each row of the coupling matrix. The block CG algorithm can then solve these systems by blocks of q rows at a time, i.e., by tackling block systems of the form $AX = B$ where now B and X are $n \times q$ matrices. Generally speaking, this improves performance by exploiting information from the solution of one linear system in the solution of another, resulting in much faster convergence[16].

Multi-grid algorithms are used to accelerate the convergence of relaxation methods like Gauss-Seidel for the numerical solution of partial differential equations. They achieve this by using a hierarchy of coarser grids with larger spacings to provide corrections to the approximate solution obtained on the finest grid. Thus, there are three parts to any multi-grid algorithm: relaxation on a given grid (also called level), restriction from a fine to a coarser grid, and interpolation back from a coarse to a finer grid[17].

As neither an efficient block CG nor an efficient multi-grid implementation were readily available, we plan to consider custom-made implementations in the future.

3.2.3 Evaluation of Coupling Matrix Elements

The final step in the construction of the coupling matrix is performing the integration of given in Eq. (15) and using the resultant $K_{ij,kl}$ to construct $Q_{ij,kl}$ according to Eq. (5). One could, in principle, use sophisticated integration algorithms to that end. However, we invariably found that for a grid spacing that was small enough to guarantee convergence of the DFT part of the calculation, direct summation was sufficiently accurate.

To optimize performance, we evaluate Eq. (15) in two steps. First, we compute the kl independent kernel of the integral:

$$\mathrm{ker}_{ij}(\vec{r}) = \Big[V_{ij}(\vec{r}) + \rho_{ij}(\vec{r})\frac{dV_{xc}[\rho(\vec{r})]}{d\rho(\vec{r})}\Big]. \tag{20}$$

This needs to be done only once per row. Furthermore, in this kernel, $\rho(\vec{r})$ is constant throughout the construction of the coupling matrix because it is set by the DFT stage. The analytic form of $V_{xc}[\rho(\vec{r})]$, and therefore $dV_{xc}[\rho(\vec{r})]/d\rho(\vec{r})$, is known. This term is therefore computed once and for all during the setup of the program. Our implementation used the Ceperley-Alder exchange-correlation functional [18]. We have slightly modified its analytic parameterization to assure a continuous derivative [19].

Once this has been calculated, computing each element of $K_{ij,kl}$ within a given row is a simple matter of computing $\psi_k(\vec{r})\psi_l^*(\vec{r})$ and summing over all grid points:

$$K_{ij,kl} = \sum_{\vec{r}} \Big[\mathrm{ker}_{ij}(\vec{r})\Big]\psi_k(\vec{r})\psi_l^*(\vec{r}). \tag{21}$$

Finally, it is straightforward to overwrite $K_{ij,kl}$ with $Q_{ij,kl}$ using Eq. (5).

Importantly, because of our use of a real-space grid, all wave functions our real and therefore both $K_{ij,kl}$ and $Q_{ij,kl}$ are inherently symmetric, so we *only compute and store their upper triangular half.*

3.3 Solving the Eigenvalue Problem

As explained in the preceding section, the eigenvalues and eigenvectors of the $Q_{ij,kl}$ matrix (see Eqs. (4)-(5)) contain all the information about the excited state properties of the system. In particular, they allow for the computation of the optical spectrum (Eq. (6)).

Fortunately, for most applications considered in this paper the $Q_{ij,kl}$ matrix is considerably smaller then the Hamiltonian matrix used for solving the DFT problem. The dimension of the Hamiltonian matrix is determined by the number of grid points, whereas the dimension of $Q_{ij,kl}$ is bound by the product of the number of filled states and the number of empty states (and can be substantially reduced further in certain applications - see the examples section below). Thus, while the Hamiltonian dimension is typically on the order of 10^5 to 10^6 for larger problems, the corresponding dimension of $Q_{ij,kl}$ is typically only 10^3 to 10^4.

For such a reasonably small matrix, finding the eigenvalues and eigenvectors can easily rely on standard subroutines. $Q_{ij,kl}$ is a symmetric, dense, and real matrix. The serial implementation of the diagnoalization used the subroutine `DSYEV` from LAPACK [20], which implements a QR factorization algorithm. After its completion, the eigenvalues and eigenvectors were used to compute the optical spectrum.

4 Parallelization

The algorithms described so far lend themselves to massive parallelization. The parallelization of the DFT part of the calculation has been described in detail previously [7, 8]. In this section, we describe how parallelism was exploited in both construction and diagonalization of the $Q_{ij,kl}$ matrix.

4.1 Coupling Matrix Construction

Because construction of each row of the coupling matrix is completely independent from the construction of other rows, parallelization with respect to rows is natural. A simple parallelization scheme would therefore use the well-known master-slave approach, where the master sets up the initial problem and each slave computes an equal number of rows. This approach is quite easy to implement, yet is very efficient as it requires no slave-slave communication.

This simple approach can still be improved substantially in terms of processor utilization and load balancing. First, because the master is only involved in setting up the problem and in storing the results, it would be idle for most of the run time. Second, because we only compute the upper triangular half of the coupling matrix, the amount of work required for each row is not constant. Third, there is no mechanism of run-time compensation for slow-down of some processors due to non-dedicated processor usage on a massively parallel platform.

We circumvented all three difficulties by adopting the following scheme for allocation of work. Two variables, *mblock* and *sblock*, were defined to contain the size of the block of work (number of rows) that should be done by the master and slave respectively. Initially they are set as:

$$sblock = max\left(\frac{qdim/nprocs}{10}, 1\right); \qquad mblock = max\left(\frac{sblock}{25}, 1\right). \tag{22}$$

where *qdim* is the total number of rows that must be computed and *nprocs* is the number of processors we are using. Every time a processor is done it is assigned another block of size *mblock* or *sblock*. If *mblock* and *sblock* were to remain constant throughout the computation, then as the end of the matrix is approached several of the processors would idle while waiting for the others to finish. Therefore, *mblock* and *sblock* are gradually "stepped down" towards the end of the matrix as follows. Before assigning work to any slave, the master checks whether there are enough rows remaining to assign every processor an *sblock* sized portion. If there isn't, *mblock* and *sblock* are decreased to:

$$sblock = mblock; \qquad mblock = max\left(\frac{mblock}{25}, 1\right). \tag{23}$$

This update takes place multiple times, until both *mblock* and *sblock* are 1. We found that this approach almost completely eliminates the idle time of the processors and provides for a self-correcting load-balancing scheme.

The implementation described so far assumed that each processor has a local copy of all wave functions so that communication with hard disk or other processors is not necessary. If memory on each processor is an issue, we have also implemented an out-of-core version, where the wave functions remain on the hard disk and are accessed as needed from each processor. Trivially, this drastically reduces the memory overhead, as the memory requirements are dominated by the wave functions. However, this comes at a significant performance penalty.

4.2 Solving the Eigenvalue Problem

This part of the computation was much less time consuming than the construction of the coupling matrix. Therefore, the need for its parallelization was not motivated by performance, but rather by memory. For large coupling matrices, stored in double precision format, it is not difficult to exhaust all memory available to a single processor. Parallelization is therefore primarily aimed at achieving a good distribution of the coupling matrix (and the resulting eigenvectors) among many processors.

As the serial implementation used the LAPACK `DSYEV` routine, the simplest parallel approach was to migrate to ScaLAPACK [21] and use the `PDSYEV` routine for diagonalization. Neglecting all other variables, this means that if M is the total memory requirement of the coupling matrix, then the memory requirement per processor is reduced to $2 \times M/nprocs$. The factor of two is because in the serial implementation it was possible to overwrite the coupling matrix array with its eigenvectors. However, this was not possible in the parallel implementation. For a reasonable *nprocs*, this is a significant reduction in memory requirements and the code is limited by computational cost and not by hardware limitations.

To avoid any unnecessary communication and calculation overlap, ScaLAPACK was not used for evaluating the oscillator strengths (Eq. (6)). Instead, the matrix-vector multiplications involved with this operation were performed using the usual master-slave paradigm.

In addition to the memory overhead reduction, the use of parallel eigenvalue solver also provided a modest performance improvement. As the performance of this part is not a bottleneck, no

further attempt at optimization was deemed necessary.

5 Examples

In this section, we show two examples of non-trivial TDLDA applications: size-dependent optical absorption in hydrogenated silicon quantum dots and optical absorption in a silicon vacancy.

5.1 Optical Absorption of Hydrogenated Silicon

We constructed quasi-spherical silicon quantum dots on the computer from "shells" of equi-distant silicon atoms situated around a central atom. The silicon atoms were placed in sites corresponding to the *bulk* structure of silicon, where each silicon atoms is surrounded by four other silicon atoms possessing tetrahedral symmetry. This results in dangling bonds for silicon atoms on the outer shells, which have "missing" neighbors. Each dangling bond was then eliminated by adding a surface hydrogen atom, thus removing optically active states associated with the surface. Finally, we relaxed the outer layers of the dot by moving the atoms so as to minimize the forces acting on the hydrogen cap atoms.

A selection of spectra obtained by computing the TDLDA spectra of the hydrogenated quantum dots is shown in Fig. 5. It starts with the smallest "quantum dot" one could construct using the above procedure - a SiH_4 molecule based on the central Si atom - and ends with $Si_{147}H_{100}$, which is the largest dot we have computed so far. For comparison, Fig. 5 also shows absorption spectra of the same dots computed using conventional (time-independent) LDA, where the spectrum is based on LDA eigenvalue differences weighted by the matrix element between the associated eigenvectors. The matrix element for the transition between eigenvalues i and j is $|(B_\beta)_{ij}|^2$ of Eq. (7) and the spectrum is averaged over the possible values of β.

Both the LDA and the TDLDA absorption spectra show an evolution with increasing dot size - from a peaked spectrum characteristic of molecules to a continuous spectrum characteristic of solids. In addition, both types of spectra feature an absorption onset that increases with decreasing dot size - the well-known "quantum size effect" [22]. The line-shape of the two types of spectra are often very different, however. Most importantly, the TDLDA absorption onset is significantly blue-shifted (shifted up in energy) with respect to the LDA one. This is to be expected, because LDA is well-known to underestimate the gap between the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO) in molecules and clusters [2].

In order to facilitate comparison to experiment, we need to define a relevant theoretical absorption onset. Ideally, the optical absorption threshold is defined as the energy of the first dipole-allowed transition. For the quantum dots studied here, there were a large number of transitions at the low-energy end of the spectrum that possessed non-zero, but very small, oscillator strengths. In experiment, such transitions would lie far below the detectability limit. We therefore defined an "experimentally observable" threshold as the energy at which the *total* oscillator strength up to that energy is a small but non-zero fraction of the overall oscillator strength. We chose the fraction to be 10^{-4}, which is above the numerical noise level but small enough to not suppress detectable dipole-allowed transitions.

A comparison between the absorption onset extracted from our calculation and several experimental studies (as a function of the quantum dot diameter, with SiH_4 considered as a "zero diameter" dot) is given in Fig. 6. It is readily observed that our calculations agree with experiment for the smallest dots and their extrapolation to larger sizes also appears to be consistent with experiment. This clearly demonstrates the strength of our approach in yielding quantitatively

accurate answers for optical properties of systems much larger than the small molecules usually attacked with TDLDA. A more detailed discussion of the physical implications of these calculations, including comparisons to other theoretical methods, can be found elsewhere [19, 23].

5.2 Optical Absorption in a Silicon Vacancy

It is well-known that point *defects* in a material, such as a missing atom, a displaced atom, or an impurity atom, can introduce new energy levels that may be optically active [24]. Recently, it has been shown that placing a point defect in a large quantum dot, to be solved in real space, can be much more effective computationally than placing it in a supercell that is to be solved in Fourier space [25]. In order to test whether our real-space TDLDA approach is sensitive enough to detect *absorption* in such a configuration, we computed the properties of a $Si_{35}H_{36}$ quantum dot possessing a vacancy. The vacancy was generated simply by removing the central silicon atom, thus obtaining a $Si_{34}H_{36}$ dot.

The removal of the central silicon atom results in new dangling bonds and disrupts the balance of forces acting on each atom. Using the LDA part of the code, we found that in the absence of atom relaxation, this removal introduces a triply degenerate energy level inside the HOMO-LUMO gap of the dot. However, it is energetically more favorable for the atoms to distort so that one juxtaposed pair of neighboring silicon atoms has a shorter bond length than the other one. This distortion, known as a Jahn-Teller distortion, gradually decreases in magnitude for subsequent shells around the vacancy [25]. It results in a split of the triply degenerate level to a singly degenerate, doubly occupied level and a doubly degenerate, unoccupied level, as shown in Fig. 7.

LDA and TDLDA optical spectra of both the vacancy-containing, relaxed $Si_{34}H_{36}$ dot and the $Si_{35}H_{36}$ dot are shown in Fig. 8a and Fig. 8b. The spectra of the two dots are practically identical above the absorption threshold of the $Si_{35}H_{36}$ dot, indicating that beyond the introduction of the gap levels the introduction of the vacancy did not perturb the electronic structure significantly. Fig. 8c shows a zoom-in on the sub-threshold region of the $Si_{34}H_{36}$ dot absorption spectrum, indicating additional, weaker, absorption features that are associated with the vacancy. Again, we find a significant difference between the predictions of LDA and TDLDA. Generally speaking, in the TDLDA spectrum the overall magnitude of the sub-bandgap features is decreased and the weight of the lower-lying transitions is diminished with respect to the higher-lying ones.

In LDA, determining whether a given sub-threshold absorption peak involves an excitation from the filled defect level or an excitation into the doubly degenerate empty defect level is trivial. All one has to do is compare the peak position with LDA occupied-unoccupied eigenvalue difference. Such straightforward assignment is not possible within TDLDA because it mixes the contribution of different LDA-level occupied-unoccupied eigenvalue pairs. However, in TDLDA the components of the eigenvector, F_I, associated with each eigenvalue, Ω_I (see Eq. 4) are indicative of the relative contribution of each LDA eigenvalue pair. Analyzing the eigenvectors therefore allows one to determine the dominant LDA-level transition that the TDLDA eigenvalue corresponds to. In the present case, such analysis has indicated that whereas in LDA the primary peak associated with an excitation into the empty defect level was at $\sim$1.7 eV, the primary TDLDA peak associated with the same transition was at $\sim$ 2.5 eV.

6 Practical Considerations

In this section, we present some practical computational considerations associated with the TDLDA examples presented in the previous section.

6.1 Computational Considerations

A significant complication associated with TDLDA is the need to consider empty states. In time-independent DFT, we only need to converge the filled states to get the right result. This is because empty states do not contribute to the charge density and therefore play no role in the self-consistent cycle. For TDLDA, however, an insufficient number of empty states or a lack of convergence of empty states would mean that the coupling matrix, and hence the entire calculation, is in error.

It is reasonable to assume (using standard perturbation theory arguments [29]) that the contribution of an individual LDA-level transition (i. e., from occupied eigenvalue i to empty eigenvalue j) to the TDLDA-level absorption at a given energy would be inversely proportional to the difference between the $i \rightarrow j$ transition energy and the given absorption energy. This means that the number of empty states one should include depends on the energy range in which the absorption spectrum is sought. An upper bound for the energy range in which the results are trustworthy is clearly the eigenvalue difference between the highest occupied state and the highest empty state. Our experience with the hydrogenated silicon quantum dots is that the spectrum is meaningful only up to a few tenths of an eV below this bound. Empirically, we found that including two to three empty states for each occupied state led to excellent convergence for the 10 eV energy range shown in Fig. 5. If one is interested primarily in the absorption threshold (as is indeed the case for the larger quantum dots), it makes little sense to compute a wide spectral range and one can use a smaller number of empty states. This was indeed done for the larger quantum dots. For example, the spectrum of $Si_{147}H_{100}$ in Fig. 5 is terminated after $\sim$7 eV.

Importantly, it is frequently the case that the lowest occupied states are rather low in energy and many $i \rightarrow j$ transitions involving them are far in access of the "trustworthy energy window" discussed above. It is therefore extremely useful to introduce a cut-off energy parameter such that $i \rightarrow j$ transitions whose transition energy exceeds the cutoff are never computed. The entire row and column involving such transitions are eliminated from the coupling matrix. We found that doing so greatly reduces the computational and memory load associated with both computing the coupling matrix and diagonalizing it, with a negligible effect on accuracy.

The empty states are much less localized than the low energy ones. Consequently, an LDA computation intended as a "pre-TDLDA" step requires not only many extra states, but also a larger sphere size to avoid a spurious effect of the zero boundary condition. For the hydrogenated silicon quantum dots, we found that a separation of at least 10 to 12 a.u. between the surface atoms and the spherical domain boundary was needed (versus a separation of 7 to 8 a.u. for a regular LDA calculation).

We recommend the following sequence of computation in order to minimize the costs associated with larger sphere and many empty states: First, compute the LDA results conventionally, using only a few empty states and a smaller sphere size. This means that the brunt of the self-consistent iterations are performed with a minimal number of eigenvalues and a smaller Hamiltonian size. Next, increase the sphere size, using the charge density from the previous step (padded with zeros outside the original sphere). Convergence will be rapid because the filled states were already quite well-described within the smaller sphere. Only then increase the number of states. Iterations to self-consistency will not be needed because the correct charge density has already been found.

Another way to mitigate the computational load associated with using a larger R is to use a coarser grid spacing then that necessary for strict convergence of the LDA wave functions. This is possible because the coupling matrix is based entirely on integrals of these wave functions and therefore dampens the high-frequency content of the wave functions.

We stress that the differences in the number of empty states, sphere size, and grid spacing for a "pre-TDLDA" and a "regular" LDA calculation mean that *parameter convergence must be tested*

on the final TDLDA spectrum and not on the LDA level results.

6.2 Scalability

Two main factors influence the scaling of the construction of the coupling matrix: the number of transitions included, $kdim$, and the number of grid points, $ndim$. The number of grid points scales roughly as n, the number of atoms in the system, but the number of transitions scales somewhere between a factor of n and of n^2, depending on the choice of the"trustworthy" energy window. The former factor is obtained if one chooses a fixed number of empty states (inexpensive, but of limited accuracy), and the latter is obtained for a fixed ratio of empty to filled states (preferable, but expensive).

As has been discussed above, the most time consuming part of the construction of the coupling matrix is the repeated solution of Eq. (13). In practice, when applied to solving Poisson's equation, it can be observed experimentally that the preconditioned CG method scales roughly as $ndim \cdot \log(ndim)$. The cost of performing the integration (14) scales as $ndim$. This integration must be performed for each entry of the coupling matrix, so the corresponding total cost scales as $kdim^2.ndim$. The CG algorithm is applied only once for each row contributing a total of order $kdim \cdot ndim \cdot \log(ndim)$ toward the overall cost. Finally the solution of the eigenvalue problem (4) scales like $kdim^3$. As a result, the total cost for computing the coupling matrix is of the form:

$$C_1 \cdot kdim^2 \cdot ndim + C_2 \cdot kdim \cdot ndim \cdot \log(ndim) + C_3 \cdot kdim^3 \tag{24}$$

where C_1, C_2, C_3 are constants.

As was already mentioned, in the best case scenario $ndim \approx \alpha n$ and $kdim \approx \beta n$, where n is the number of atoms. In this case, the total cost scales like

$$(C_3\beta + C_1\alpha)\beta^2 \cdot n^3 + C_2\alpha\beta \cdot n^2 \log(\alpha n). \tag{25}$$

In the worst case scenario $kdim \approx \beta n^2$, in which case the total cost will scale like

$$\left(C_3\beta + C_1\frac{\alpha}{n}\right)\beta^2 \cdot n^6 + C_2\alpha\beta \cdot n^3 \log(\alpha n). \tag{26}$$

In practice, $kdim$ lies between n and n^2 and the above mentioned techniques can be used to keep this as close to n as possible. However, focusing on the best case scenario (25), one cannot ignore the relative size of the prefactors $(C_3\beta + C_1\alpha)\beta^2$ and $C_2\alpha\beta$. For systems such as $Si_{34}H_{36}$, the computational overhead is dominated almost entirely by the CG term on the right. For larger systems it can be expected that the left hand term will begin to dominate, but it is not clear at exactly what point this will happen. Based on our observations of calculations for $Si_{34}H_{36}$, where $n = 70$, we would estimate that the ratio of these prefactors is in the range of 10^4 to 10^5. This is based on the profiling of computations showing that CG calculations accounted for approximately 70% to 80% of the total runtime. As a result, in the best case scenario of Eq. (25) we estimate that the computational cost of the CG method will stop dominating the overall calculation when $n \sim 800$. The break-even point will take place much earlier for the worst case scenario of Eq. (26).

The large scaling factors for the creation of the coupling matrix underscore the need for good optimization and efficient parallelization. As an example of this importance, Table (1) shows the runtime for the coupling matrix generation for $Si_{34}H_{36}$ at various stages of optimization. Ultimately, we were able to reduce the runtime by almost a factor of four, a significant improvement which has allowed us to consider larger systems than were previously possible.

The current version of the coupling matrix construction code is very efficient. To show this we present Fig. 9. This figure shows the wall-clock runtime on various numbers of processors and the ideal runtime, calculated by dividing the single processor runtime by the number of processors used. Ideally, this number would be equal to the runtime of the single processor case divided by the number of processors used. We can see that the runtime is actually better than ideal for small numbers of processors. This is because disk reading and writing is handled by the master processor, so in the single processor case no work is being done during output. In other cases the slave processors continue to work while the master writes the coupling matrix.

Until now the largest system that we have investigated is $Si_{147}H_{100}$. Based on the scaling arguments presented above, the computation of a $Si_{275}H_{172}$ dot (at the bottom end of the experimental range for the larger quantum dots shown in Fig. 6) would be larger than that of the $Si_{147}H_{100}$ dot by at least half an order of magnitude, if not a full order of magnitude. Due to limited computer time, we have not performed that calculation yet. However, it is important to note that before optimization of the code computing a system of this size would have been outright impossible. We hope to bridge the gap between theory and experiment for the data of Fig. 6 in future work.

7 Conclusion

In conclusion, we have presented a detailed overview of a massively parallel implementation of a time-dependent density functional theory code aimed at computing optical absorption spectra of systems with hundreds of atoms. We have shown that using the formalism of Casida [4] explicit time-dependent computation is avoided by formulating an eigenvalue problem in frequency space, that is constructed from the solutions of the time-independent density functional formulation. We have presented details of an efficient implementation of this formulation in real space, using norm conserving pseudopotentials and a higher order finite difference approach. The bottleneck of this implementation lies in the repeated solution of the Poisson equation. We discussed various strategies for optimizing this calculation. We have also shown that our implementation lends itself naturally to massive "master-slave" parallelization that does not involve any slave-slave or global communication steps. Load balancing optimization, memory/run-time tradeoffs, and scalability issues were discussed. We have illustrated the capabilities of the code by presenting calculations for the quantum size effect and for absorption in defects in hydrogenated silicon quantum dots with as many as $\sim$450 atoms and have outlined how one should proceed with even larger computations.

Acknowledgements

We thank Zhongze Li for his help in parallelizing an early version of the coupling matrix construction code, and Yu Liang for his assistance in adapting the Incomplete Cholesky factorization preconditioning for use in this work. LK acknowledges the generous support of the Estelle Funk Foundation and the Delta Career Development Chair. We acknowledge support for this work by the National Science Foundation (under Grant ITR 0082094) and the Minnesota Supercomputing Institute.

Appendix A: Multipole Expansion

The ℓ^{th} term of the multipole expansion of Eq. (16) at a general point with spherical coordinates of (R, θ, ϕ) is given by [30]:

$$\Phi^{(\ell)} = \frac{1}{R^{\ell+1}} \sum_{m=-\ell}^{\ell} \sqrt{\frac{4\pi}{2\ell+1}} Q_m^{(\ell)} Y_{\ell,m}^*(\theta,\phi). \tag{A1}$$

In Eq. (A1), ℓ is a non-negative integer. $Y_{\ell,m}(\theta,\phi)$ is the spherical harmonic evaluated at the general point and is given by:

$$Y_{\ell,m}(\theta,\phi) = \sqrt{\frac{2\ell+1}{4\pi}\frac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(cos(\theta)) e^{im\phi}, \tag{A2}$$

where $P_\ell^m(x)$ are the associated Legendre polynomials, defined by:

$$P_\ell^m(x) = \frac{(-1)^m}{2^\ell \ell!}(1-x^2)^{m/2}\frac{d^{\ell+m}}{dx^{\ell+m}}(x^2-1)^\ell. \tag{A3}$$

$Q_m^{(\ell)}$ is calculated from the distribution of point charges that generate the potential as [11]:

$$Q_m^{(\ell)} = \sum_a e_a r_a^\ell \sqrt{\frac{4\pi}{2\ell+1}} Y_{\ell,m}(\theta_a,\phi_a), \tag{A4}$$

where e_a is a point charge situated at the spherical coordinates (r_a,θ_a,ϕ_a).

Spherical harmonics have the additional property that [11]:

$$Y_{\ell,-m}(\theta,\phi) = (-1)^m Y_{\ell,m}^*(\theta,\phi). \tag{A5}$$

This property can be used to reconstruct Eq. (A1) so as to sum over non-negative m values only. Using Eq. (A5) in Eq. (A4) yields:

$$Q_{-m}^{(\ell)} = (-1)^m Q_m^{(\ell)*} \tag{A6}$$

Combining Eqs. (A5) and (A6) shows that each negative m term in Eq. (A1) is the complex conjugate of the corresponding positive m term in the equation. Therefore, Eq. (A1) can be written as:

$$\Phi^{(\ell)} = \frac{1}{R^{\ell+1}}\sqrt{\frac{4\pi}{2\ell+1}}(Q_0^{(\ell)} Y_{\ell,0}(\theta,\phi) + \sum_{m=1}^{\ell} Q_m^{(\ell)} Y_{\ell,m}^*(\theta,\phi)). \tag{A7}$$

Using Eqs. (A2)-(A4) in Eq. (A7) we obtain our final expression:

$$\begin{aligned}\Phi^{(\ell)} &= \frac{1}{R^{\ell+1}}\Bigg[\sum_a \Big[e_a r_a^\ell P_\ell^0(cos(\theta_a))\Big] P_\ell^0(cos(\theta)) \\ &\quad +2\cdot \mathrm{Re}\Bigg\{\sum_{m=1}^{\ell}\Bigg[\frac{(\ell-m)!}{(\ell+m)!}\sum_a \Big[e_a r_a^\ell P_\ell^m(cos(\theta_a)) e^{im\phi_a}\Big] P_\ell^m(cos(\theta)) e^{-im\phi}\Bigg]\Bigg\}.\end{aligned} \tag{A8}$$

References

[1] See, e. g., J. R. Chelikowsky and S. G. Louie (Eds.), *Quantum Theory of Real Materials* (Kluwer, Boston, 1996); K. Ohno, E. Esfarjani, and Y Kawazoe, *Computational Materials Science* (Springer-Verlag, Berlin, 1999).

[2] See, e. g., M. L. Cohen and J. R. Chelikowsky, *Electronic Structure and Optical Properties of Semiconductors*, 2nd ed. (Springer-Verlag, Berlin 1989); J. R. Chelikowsky and M. L. Cohen, in *Handbook on Semiconductors*, T. S. Moss, Ed. , 2nd Edition (Elsevier, Amsterdam, 1992); W. E. Pickett, Comput. Phys. Reports **9**, 115 (1989); M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopulos, Rev. Mod. Phys. **64**, 1045 (1992).

[3] E. K. U. Gross, J. F. Dobson, and M. Petersilka, in *Density Functional Theory*, edited by R. F. Nalewajski, (Springer-Verlag, Berlin, 1996), p. 81; M. Petersilka, U. J. Gossmann, and E. K. U. Gross, Phys. Rev. Lett. **76**, 1212 (1996).

[4] M. E. Casida, in *Recent Advances in Density-Functional Methods*, Part I, edited by D. P. Chong (World Scientific, Singapore, 1995), p. 155; M. E. Casida, in *Recent Developments and Applications of Modern Density Functional Theory*, edited by J. M. Seminario (Elsevier, Amsterdam, 1996), p. 391.

[5] J. R. Chelikowsky, N. Troullier, and Y. Saad, Phys. Rev. Lett **72**, 1240 (1994); J. R. Chelikowsky, N. Troullier, K. Wu, and Y. Saad, Phys. Rev. **B50**, 11355 (1994).

[6] L. Kronik, I. Vasiliev, M. Jain, and J. R. Chelikowsky, J. Chem. Phys. **115**, 4322 (2001).

[7] Y. Saad, A. Stathoupolos, J. R. Chelikowsky, K. Wu, and S. Öğüt, BIT **36**, 563 (1996).

[8] A. Stathopoulos, S. Öğüt, Y. Saad, J. R. Chelikowsky, and H. Kim, Comput. Sci. Eng., **2**, 19 (2000).

[9] G. D. Smith, *Numerical Solutions of Partial Differential Equation: Finite Difference Methods*, 2nd ed. (Oxford, New York, 1978).

[10] J. W. Demmel, *Applied Numerical Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, 1997)

[11] J. D. Jackson, Classical Electrodynamics, 2nd edition (Wiley, New York, 1975), pp. 98-100.

[12] Y. Saad, *SPARSKIT: A basic tool kit for sparse matrix computations*, Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

[13] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, SIAM (2003), see also: http://www-users.cs.umn.edu/ saad/books.html.

[14] See, e. g., J. C. Diaz, K. Komara, *Incomplete Multilevel Cholesky Factorizations*, SIAM J. Matrix Analysis Appl., **22**, 3, 895 (2000); M. Jones, P. Plassman, *An Improved Cholesky Factorization*, ACM TOMS, **21**, 5, (1995).

[15] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computations, **31** (1977), pp. 148-162.

[16] D. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra and its Applications, **29** (1980), pp. 243-322.

[17] W. Hackbusch, *Multi-Grid Methods and Applications* (Springer, New York, 1986)

[18] D. M. Ceperley and B. J. Alder, Phys. Rev. Lett. **45**, 566 (1980).

[19] I. Vasiliev, S. Öğüt, and J. R. Chelikowsky, Phys. Rev. B **65**, 115416 (2002).

[20] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. *LAPACK Users Guide*, 3rd Edition (Society for Industrial and Applied Mathematics, Philadelphia, 1999).

[21] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users Guide* (Society for Industrial and Applied Mathematics, Philadelphia, 1997).

[22] See, e. g., A. D. Yoffe, *Adv. Phys.* **42**, 173 (1993).

[23] I. Vasiliev, S. Öğüt, and J. R. Chelikowsky, Phys. Rev. Lett **86**, 1813 (2001).

[24] M. Lanoo and J. Bourgoin, Point Defects in Semiconductors, Vols. 1 and 2 (Springer, Berlin, 1981).

[25] S. Öğüt, H. Kim, and J. R. Chelikowsky, Phys. Rev. **B56**, R11353 (1997); S. Öğüt and J. R. Chelikowsky, Phys. Rev. Lett **83**, 3852 (1999).

[26] U. Itoh, Y. Toyoshima, H. Onuki, N. Washida, and T. Ibuki, J. Chem. Phys. **85**, 4867 (1986).

[27] S. Furukawa and T. Miyasato, Phys. Rev. B **38**, 5726 (1988); D. J. Lockwood, A. Wang, and B. Bryskiewicz, Solid State Comm., **89**, 587 (1994).

[28] B. Delley and E. F. Steigmeier, Phys. Rev. B **47**, 1397 (1993).

[29] L. D. Landau and E. M. Lifshitz, *Quantum Mechanics*, 3rd edition (Pergamon Press, Oxford, 1977).

[30] L. D. Landau and E. M. Lifshitz, the classical theory of fields, 4th edition (Pergamon, Oxford, 1975), pp. 97-99.

Version	Wall-Clock Run Time (hours)
Initial Parallel Implementation	53
Optimized Load Balancing	24
Memory Management and Calculation Overlap Optimization	20
Preconditioned CG Optimization	15

Table 1: Wall-clock runtime of the parallel TDLDA code (*not* including the generation of Kohn-Sham eigenvalues and wave functions) for the $Si_{34}H_{36}$ test case running on 8 processors at various stages in the optimization process.

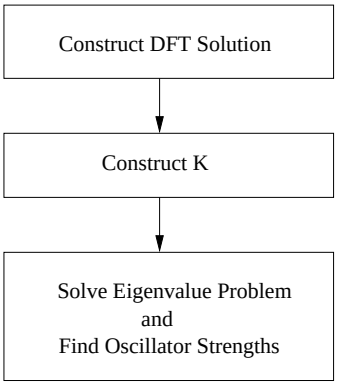

Figure 1: Flowchart showing major parts of the code.

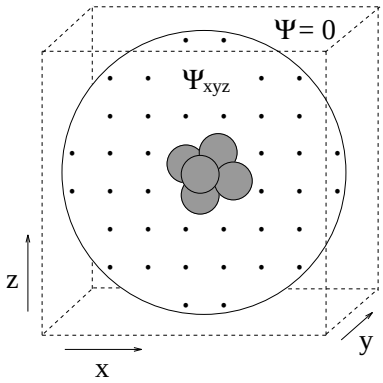

Figure 2: Schematic illustration of the real-space mapping used in the code.

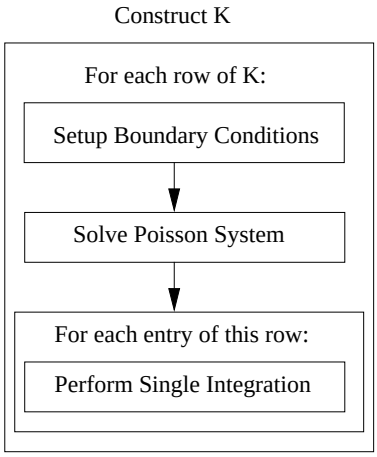

Figure 3: Flowchart showing details of matrix construction.

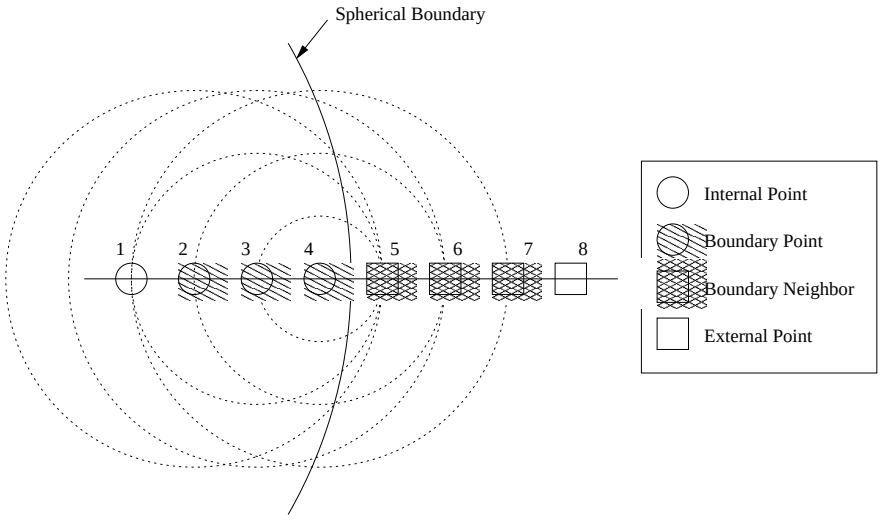

Figure 4: One dimensional diagram showing the different types of grid points. The dotted circles about each boundary point help to indicate which external points are its boundary neighbors.

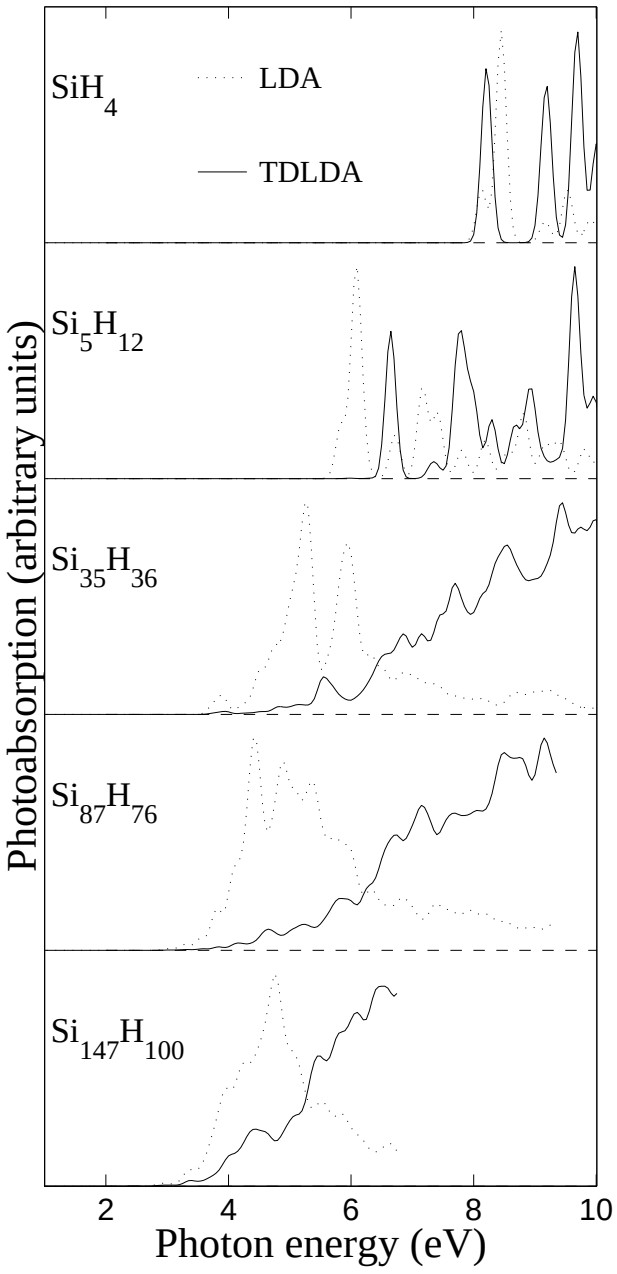

Figure 5: LDA and TDLDA absorption spectra of selected hydrogenated silicon quantum dots.

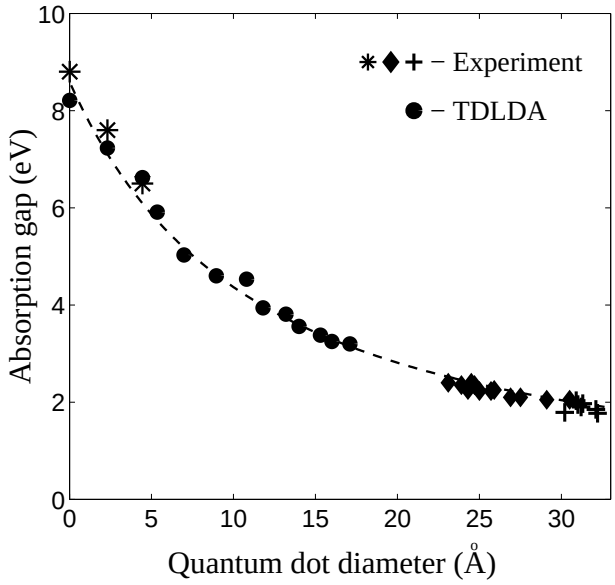

Figure 6: Variation of the optical absorption threshold of hydrogenated silicon quantum dots as a function of the dot diameter. Circles - TDLDA calculation. Experimental values are taken from refs. [26, 27, 28]. The dashed line is intended only as a guide to the eye.

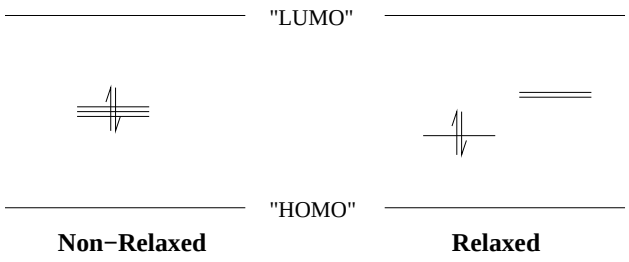

Figure 7: Schematic depiction of level splitting in a relaxed and a non-relaxed $Si_{34}H_{36}$ quantum dot.

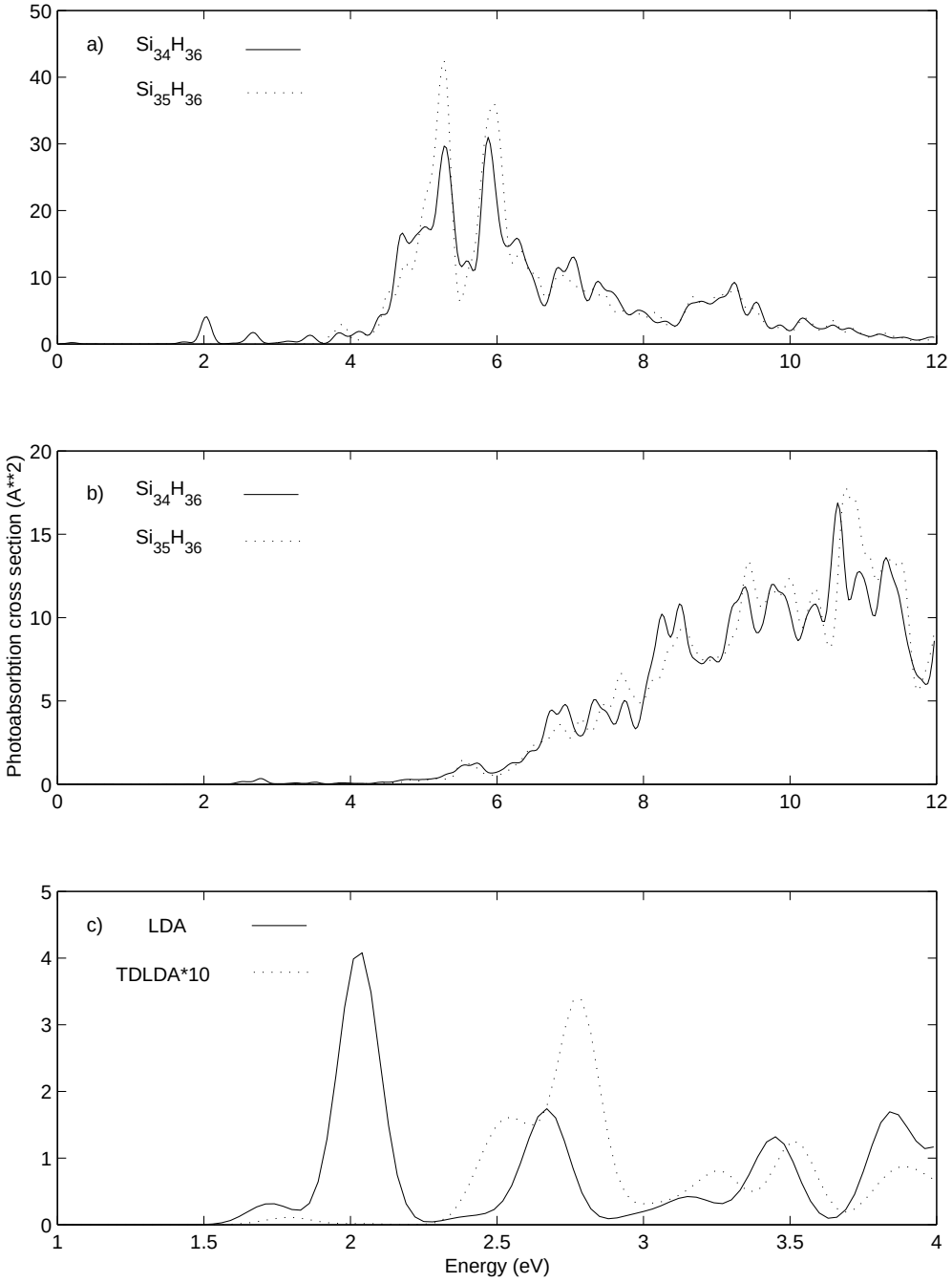

Figure 8: (a) LDA and (b) TDLDA optical spectra for $Si_{35}H_{36}$ and $Si_{34}H_{36}$ quantum dots. (c) Sub-threshold portion of the LDA and TDLDA optical spectra of the $Si_{34}H_{36}$ quantum dot.

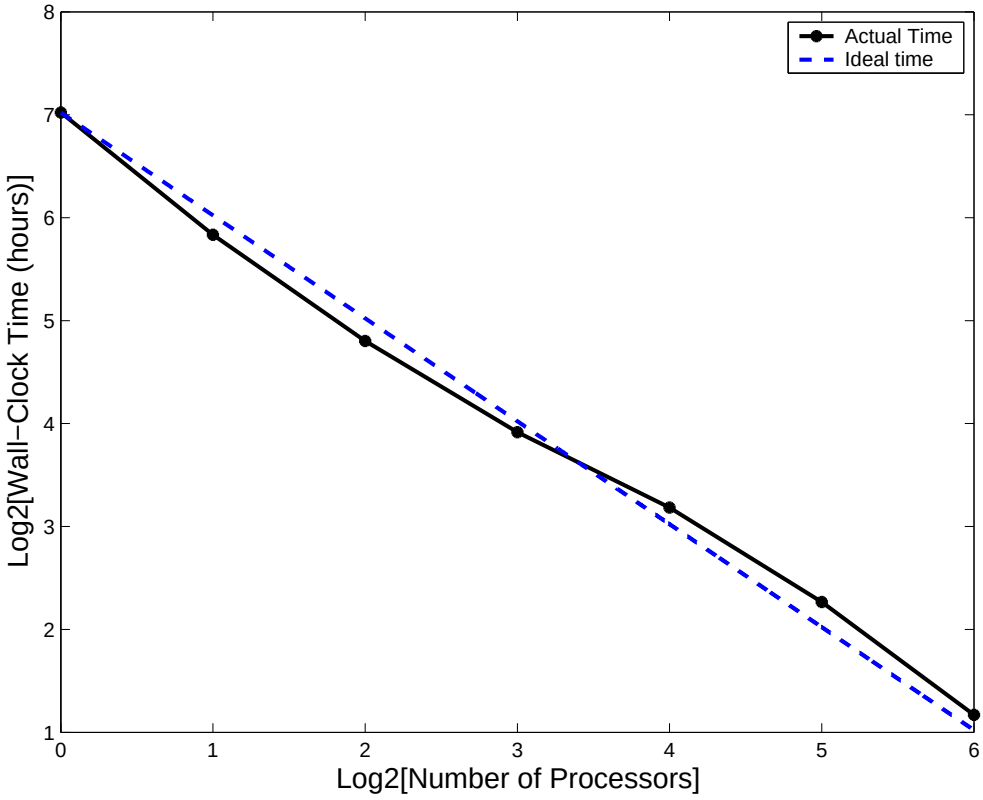

Figure 9: Scalability of coupling matrix construction based on wall-clock runtime.