CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–27
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Design and performance characterization of electronic structure calculations on massively parallel supercomputers: A case study of GPAW on the Blue Gene/P architecture

N. A. Romero,[1]*C. Glinsvad,[2] A. H. Larsen,[3,4] J. Enkovaara,[5,6] S. Shende,[7] V. A. Morozov,[1] and J. J. Mortensen,[3]

[1]*Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439, USA*
[2]*Center for Individual Nanoparticle Functionality, Department of Physics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark*
[3]*Center for Atomic-scale Materials Design, Department of Physics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark*
[4]*Nano-bio Spectroscopy Group and ETSF Scientific Development Centre, Departamento de Física de Materiales, Universidad del País Vasco, CSIC–UPV/EHU–MPC and DIPC, Avenida de Tolosa 72, E-20018 San Sebastián, Spain*
[5]*CSC - IT Center for Science Ltd., P.O. Box 405 FI-02101 Espoo, Finland*
[6]*Department of Applied Physics, School of Science, Aalto University, P.O. Box 11100 FI-00076 AALTO, Finland*
[7]*Performance Research Laboratory, University of Oregon, Eugene, OR, 97403, USA*

SUMMARY

Density function theory (DFT) is the most widely employed electronic structure method due to its favorable scaling with system size and accuracy for a broad range of molecular and condensed-phase systems. The advent of massively parallel supercomputers has enhanced the scientific community's ability to study larger system sizes. Ground-state DFT calculations on $\sim 10^3$ valence electrons using traditional $\mathcal{O}(N^3)$ algorithms can be routinely performed on present-day supercomputers. The performance characteristics of these massively parallel DFT codes on $>10^4$ computer cores are not well understood. The GPAW code was ported an optimized for the Blue Gene/P architecture. We present our algorithmic parallelization strategy and interpret the results for a number of benchmark test cases.

Received . . .

KEY WORDS: GPAW, electronic structure, DFT, Blue Gene, massive parallelization, high-performance computing

1. INTRODUCTION

Kohn–Sham density functional theory (DFT) [1, 2] continues to play an important role in the computational modeling of molecules and condensed-phase systems. Its popularity is due to a number of factors, such as accuracy over a wide range of systems (e.g., insulators, metals, molecules), its first-principles nature, and the ability to treat relevant system sizes on commodity Linux clusters. The performance of DFT software packages is not well understood by the community at large. In an era of rapidly evolving high-performance computing (HPC) architectures, the design and performance characteristics of massively parallel DFT codes is of great importance because it is a means to enabling high-impact science.

*Correspondence to: naromero@alcf.anl.gov

Typical DFT calculations include geometry optimization and Born–Oppenheimer molecular dynamics (BOMD). These types of calculations require $\sim 10^{3-6}$ electronic minimization steps and thus large amounts of computer time. Even with multiple levels of parallelization and large supercomputers, the time-to-solution[†] for these calculations can be on the order of several days. Strong-scaling performance bottlenecks are present over a wide range of system sizes relevant to users due to serial parts (or parts with limited parallelization) in traditional $\mathcal{O}(N^3)$ DFT algorithms, a behavior known as Amdahl's Law [3].

GPAW [4, 5] is a real-space finite-difference (FD) [6, 7] DFT code based on the projector augmented-wave (PAW) [8] method. In addition to standard ground-state DFT, extensions such as time-dependent DFT [9, 10], GW approximation [11], and Bethe-Salpeter equation [12] are implemented in GPAW. It is also possible to use localized atomic orbitals (LCAOs) [13, 14], which are complementary to real-space grids, and plane wave basis sets. In this work, we focus on the parallelization of real-space ground-state DFT calculations.

GPAW is written in the Python and C programming languages [15] and uses the MPI [16, 17] programming model for parallel execution. Originally developed on commodity Linux clusters, GPAW has now been successfully ported for use on Blue Gene/P [18] (and other massively parallel architectures such as the Cray XT series) on $>10^5$ computational cores. From the beginning, GPAW has been designed for large-scale scalability, and it currently supports multiple levels of parallelization. Here, we investigate the performance of the two main non-trivial levels of parallelization: real-space domain decomposition and band parallelization. We note that this two-level parallelization approach is also implemented in at least one other real-space DFT code JuRS [19], as well as a number of plane wave based DFT codes including ABINIT [20], CPMD [21, 22], Qbox [23], Quantum ESPRESSO [24], NWChem [25], PEToT [26], and VASP [27].

Section 2 describes the algorithm for minimizing the Kohn–Sham energy functional, and Section 3 is an in-depth description of the parallelization strategy employed. Performance bottlenecks based on Amdahl's Law are derived by simple analytical considerations. Computational results for various system sizes are presented and analyzed in Section 4. Finally, a summary and an outlook are provided in Section 5.

2. MINIMIZING THE KOHN–SHAM ENERGY FUNCTIONAL

There are two general families of methods for calculating the Kohn–Sham (KS) ground state: (i) Direct minimization methods and (ii) iterative methods where the KS Hamiltonian is diagonalized in conjunction with density mixing. Iterative diagonalization methods contain two levels of iteration, one for the eigenvalue problem and an another one for the density, and they are often referred to as self-consistent field (SCF) methods [28]. GPAW employs SCF methods to obtain the KS ground state and offers three different iterative eigensolvers: (i) residual minimization method by direct inversion in the iterative subspace (RMM-DIIS) [29, 28], (ii) conjugate gradient method [30, 31], and (iii) Davidson's method [32, 28]. The RMM-DIIS method is the main iterative eigensolver and has been optimized for maximum scalability on high-performance computing platforms, and is thus our main focus in this work. A detailed description of the PAW method as implemented in the GPAW code is found in Reference [5]; only the salient features are presented here.

Within the PAW approximation, the SCF method for minimizing the KS energy functional leads to the generalized eigenvalue equation for the pseudo (PS) wave functions $\tilde{\psi}_n$,

$$\hat{H}\tilde{\psi}_{skn} = \epsilon_{skn}\hat{S}\tilde{\psi}_{skn}, \tag{1}$$

where s is the spin index, k the k-point index, and n the band index. We limit our discussion mostly to spin-unpolarized systems with a single k-point (Γ-point sampling of the Brillouin zone), and thus the s and k indices are omitted. The Hamiltonian operator $\hat{H}$ is nonlinear as it depends on the wave functions. The $\hat{S}$ is the overlap operator, which can be defined in terms of the PAW transformation

[†]The total wall-clock time from beginning to end of the calculation.

DOI: 10.1002/cpe

operator $\hat{\mathcal{T}}$ such that

$$\hat{S} = \hat{\mathcal{T}}^\dagger \hat{\mathcal{T}} = 1 + \sum_a \sum_{i_1 i_2} |\tilde{p}^a_{i_1}\rangle \Delta S^a_{i_1 i_2} \langle \tilde{p}^a_{i_2}|. \tag{2}$$

Here a is an atomic index and i_1, i_2 are combination indices for the principal, angular momentum, and magnetic quantum numbers. The projector functions $\tilde{p}^a_i$ are localized to an augmentation sphere and $\Delta S^a_{i_1 i_2}$ are the atomic corrections to the overlap operator.

The Hamiltonian operator in Equation 1 is given by

$$\hat{H} = -\frac{1}{2}\nabla^2 + \tilde{v} + \sum_a \sum_{i_1 i_2} |\tilde{p}^a_{i_1}\rangle \Delta H^a_{i_1 i_2} \langle \tilde{p}^a_{i_2}|, \tag{3}$$

where $\Delta H^a_{i_1 i_2}$ represents the atomic corrections and the effective potential is

$$\tilde{v} = \tilde{v}_{\text{coul}} + \tilde{v}_{\text{xc}} + \sum_a \bar{v}^a. \tag{4}$$

The Coulomb potential $\tilde{v}_{\text{coul}}$ satisfies the Poisson equation for the charge density (including the compensation charge contribution) $\nabla^2 \tilde{v}_{\text{coul}} = -4\pi\tilde{\rho}$, while $\tilde{v}_{\text{xc}}$ is the exchange–correlation (XC) potential and $\bar{v}^a$ represents fixed local potentials around each atom.

The SCF solution of Equation 1 using the RMM-DIIS algorithm proceeds as follows:

1. Initial guess for PS wave functions $\tilde{\psi}_n$ given by the LCAO initialization.
2. Orthogonalization of wave functions:
   (a) Construction of overlap matrix: $S_{mn} = \langle \tilde{\psi}_m | \hat{S} | \tilde{\psi}_n \rangle$.
   (b) Cholesky factorization of the overlap matrix: $S = L^T L$.
   (c) Rotation of wave functions: $L^{-1}\tilde{\psi}_n \to \tilde{\psi}_n$.
3. Calculation of the PS density: $\tilde{n}(\mathbf{r}) = \sum_n f_n |\tilde{\psi}_n(\mathbf{r})|^2 + \sum_a \tilde{n}^a_c(\mathbf{r})$, where f_n represents the occupation numbers and $\tilde{n}^a_c$ is a smooth PS core density equal to the all-electron (AE) core density outside the augmentation sphere.
4. Calculation of the Coulomb and effective potential: $\tilde{v}_{\text{coul}}, \tilde{v}$.
5. Application of the KS operator on the PS wave functions: $\hat{H}\tilde{\psi}_n$.
6. Subspace diagonalization:
   (a) Construction of the Hamiltonian matrix: $H_{mn} = \langle \tilde{\psi}_m | \hat{H} | \tilde{\psi}_n \rangle$.
   (b) Diagonalization of the Hamiltonian matrix gives the eigenvalues and eigenvectors in the subspace: ϵ_n, W_{mn}.
   (c) Rotation of wave function: $W\tilde{\psi}_n \to \tilde{\psi}_n$.
7. Update of wave functions by residual minimization:
   (a) Calculation of residuals: $R_n = (\hat{H} - \epsilon_n \hat{S})\tilde{\psi}_n$.
   (b) Improvement of the PS wave functions: $\tilde{\psi}'_n = \tilde{\psi}_n + \lambda \hat{P} R_n$, where the preconditioner is the approximate inverse of the kinetic energy operator $\hat{T}$, $\hat{P} = \hat{T}^{-1}$. The step length is chosen to minimize the norm of the residual for the new guess: $R'_n = (\hat{H} - \epsilon_n \hat{S})\tilde{\psi}'_n$.
   (c) Final trial step with the same step length: $\tilde{\psi}'_n = \tilde{\psi}_n + \lambda \hat{P} R_n + \lambda \hat{P} R'_n$.

   The algorithm is equivalent to RMM-DIIS as described in Ref. [28] with the subspace dimension of two.
8. If energy, density, and wave functions have not converged, return to Step 2; otherwise, the process is done.

The equations are discretized by representing the wave functions, densities, and other variables with their values at points in a uniform real-space grid; for example, $\tilde{\psi}_n(\mathbf{r}) = \tilde{\psi}_{ng}$ where g is a combination index for the grid points in the Cartesian dimensions. In the discretized form the Hamiltonian can be written as

$$H_{gg'} = -\frac{1}{2} L_{gg'} + \tilde{v}_g \delta_{gg'} + \sum_{i_1 i_2} \tilde{p}^a_{i_1 g} \Delta H^a_{i_1 i_2} \tilde{p}^a_{i_2 g'} \tag{5}$$

DOI: 10.1002/cpe

where $L_{gg'}$ is the finite-difference stencil for the Laplacian. The Hamiltonian is sparse, and because the RMM-DIIS algorithm requires only evaluation of matrix-vector products, the full matrix is never stored.

3. PARALLELIZATION STRATEGY

The parallelization strategy employed in GPAW is co-designed with the memory requirements, floating point operations, and communication patterns of the RMM-DIIS algorithm. These resource requirements are related to three extensive parameters characterizing the size of the calculation: (i) total number of bands N_b (or states in the case of finite systems),[‡] (ii) total number of grid points N_g, and (iii) total number of PAW projectors N_p, which is related to the total number of atoms N_a and their chemical species. For real physical systems, these parameters are inter-related:

- $N_g \gg N_b$, which is the mathematical condition motivating iterative diagonalization instead of direct diagonalization.
- $N_p \geq N_a$, since each atom generally has at least one projector function for each valence state.

The goal of the parallelization strategy presented here is to achieve the best weak- and strong-scaling performance. Weak-scaling is primarily concerned with enabling calculations on larger system sizes, which might otherwise not be possible due to hardware constraints (e.g., memory), while strong-scaling is primarily concerned with reducing the time-to-solution for a fixed system size. The performance of a DFT code is determined by a number of aspects that cross-cut hardware and software, as summarized here:

1. Initialization phase (which includes reading the input file and PAW pseudopotentials, as well as the LCAO calculation which generates the initial PS wave functions);
2. Scalable data structures;
3. Amdahl's Law bottlenecks in the main SCF algorithm;
4. Scalable communication patterns;
5. Single-core peak performance of key algorithmic kernels; and
6. Check-pointing (with a frequency determined by the user).

In this paper we focus on the Items 2 through 4 from this list in Sections 3.1–3.3. We briefly touch upon the remaining items in Section 3.4.

3.1. Memory and Matrix Distribution

The first hardware resource that becomes exhausted as the system size increases is the memory per node. The matrix requiring the largest amount of memory stores the PS wave functions. The PS wave functions $\tilde{\psi}_{ng}$ are stored naturally as a four-dimensional matrix (as g is a combination index for the three Cartesian dimensions). The storage requirement for $\tilde{\psi}_{ng}$ is thus $\mathcal{O}(N_b N_g)$. Beyond a certain system size, it becomes necessary to distribute $\tilde{\psi}_{ng}$ along the band index n and grid index g *simultaneously*. Figure 1 illustrates the simplicity behind this approach, which is inspired by well-known concepts in the parallel dense linear algebra community [33, 34, 35] and is now employed in several DFT codes [20, 21, 22, 23, 24, 25, 26, 27]. The PS wave functions $\tilde{\psi}_{ng}$ are distributed on a $B \times G$ process grid, where B and G are the total number of band groups and real-space domains, respectively. Note that G is the product of G_1, G_2, and G_3, the domain decomposition along each of the three axes of the unit cell. Thus in our parallelization strategy, there are a total of $B \times G$ MPI tasks, with each task assigned a single contiguous range of rows and columns of $\tilde{\psi}_{ng}$ (this layout is commonly known as a two-dimensional block layout). For algorithmic simplicity, we require

[‡]Note that N_b is not to be confused with N_{val}, the total number of valence electrons. In direct minimization methods, N_b is typically equal to $N_{\text{val}}/2$, which is the number of occupied bands, while iterative diagonalization methods, including RMM-DIIS, require extra unoccupied bands to achieve convergence, hence $N_b > N_{\text{val}}/2$.

DOI: 10.1002/cpe

$N_b \bmod B = 0$ but allow $N_{g_i} \bmod G_i \neq 0$ for $i \in 1, 2, 3$ since doing otherwise would lead to changes in the grid spacing as a function of the number of MPI tasks, which is undesirable.

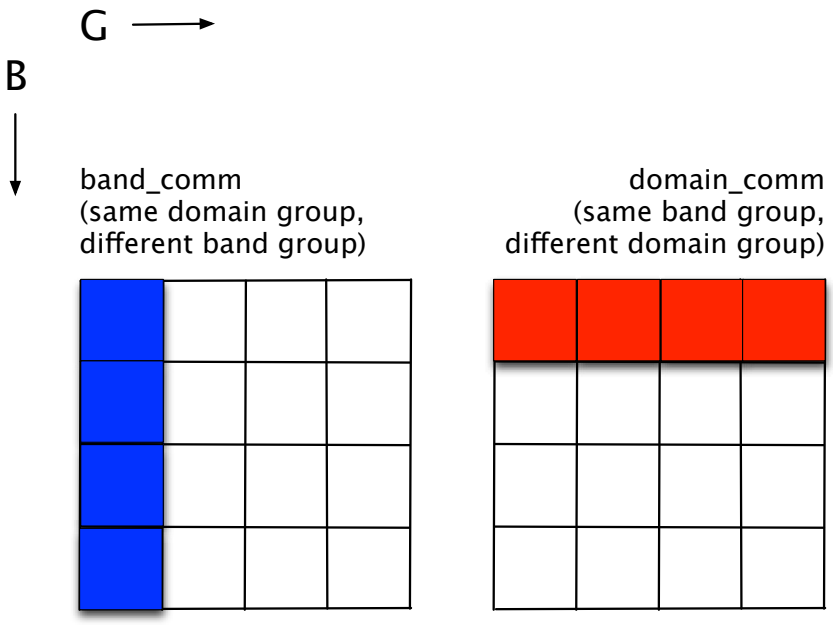


Figure 1. Two-dimensional grid layout of PS wave functions. Each square represents an MPI task and belongs to two MPI communicators: band_comm (blue) and domain_comm (red). Each MPI task is assigned a single contiguous slice of $\tilde{\psi}_{ng}$; band_comm contains MPI tasks with different n indices but the same g indices; domain_comm contains MPI tasks with different g indices but the same n indices.

The memory requirements for a spin-unpolarized Γ-point calculation are shown in Table I. Note that both the order of memory storage as well as the distribution vary considerably among the different matrices. Although at first glance it would seem natural to distribute all matrices across the entire $B \times G$ process grid, this turns out to be impractical because it would lead to an increase in communication among the MPI tasks. Thus the matrices in GPAW are either completely replicated, partially replicated, or fully distributed with respect to the $B \times G$ process grid.

Table I. Memory requirements for the RMM-DIIS algorithm. Order of memory storage for matrices as function of extensive system parameters and distribution with respect to the $B \times G$ process grid.

Matrix	Memory Storage	B-axis	G-axis
$\tilde{\psi}_{ng}$	$\mathcal{O}(N_b N_g)$	distributed	distributed
$\tilde{p}_i^a(\mathbf{r})$	$\mathcal{O}(N_p)$	replicated	distributed
$\tilde{n}(\mathbf{r}), \tilde{v}(\mathbf{r}), \tilde{v}_{\text{coul}}(\mathbf{r})$	$\mathcal{O}(N_g)$	replicated	distributed
H_{mn}, S_{mn}, W_{mn},	$\mathcal{O}(N_b^2)$	distributed	replicated
neighbor list	$\mathcal{O}(N_a)$	replicated	replicated

While the PS wave functions are fully distributed along the B and G-axes, the real-space PAW projectors are distributed along the G-axis and are partially replicated along the B-axis. The PAW projectors are evaluated in real-space and only exist inside an atom-centered augmentation sphere. This has the advantage of requiring only $\mathcal{O}(N_p)$ storage instead of the $\mathcal{O}(N_p N_g)$ storage that is needed in plane wave DFT codes using reciprocal-space projectors. There are also a number of

DOI: 10.1002/cpe

matrices (e.g., $\tilde{n}(\mathbf{r})$) that require $\mathcal{O}(N_g)$ memory storage and are distributed along the G-axis and replicated along B-axis.

The Hamiltonian H_{mn} and overlap S_{mn} matrices in the subspace of computed bands require $\mathcal{O}(N_b^2)$ storage. These matrices are *not* frequently distributed in DFT codes since they are negligible in size for most problems. Clearly, as the system size N_b increases, H_{mn} and S_{mn} will eventually need to be distributed across multiple nodes. H_{mn} and S_{mn} are distributed only along the B-axis (one-dimensional block layout) and are replicated across the G-axis, effectively slicing these matrices along the row or column index but not both. Although these matrices are two-dimensional, a one-dimensional distribution has not proven to be a memory bottleneck for problems up to $N_b \sim 10^4$. Lastly, there are $\mathcal{O}(N_a)$ arrays that are completely replicated since they require modest amounts of memory even for $N_a \sim 10^3$, which is near the practical limits of $\mathcal{O}(N^3)$ DFT methods.

3.2. Floating Point Operations and Amdahl's Law

The strong-scaling efficiency of a parallel code is limited by the least parallel component of the algorithm. We identify these Amdahl's Law bottlenecks by analyzing GPAW's RMM-DIIS algorithm, as well as confirming these results with computational experiments. Table II is a breakdown of the computational complexity for the individual steps in the SCF cycle. Although DFT methods are often associated with having $\mathcal{O}(N^3)$ computational complexity, the full description of the complexity contains five types of terms (in decreasing order): (i) $\mathcal{O}(N_b^2 N_g)$, (ii) $\mathcal{O}(N_b^3)$, (iii) $\mathcal{O}(N_b N_g)$, (iv) $\mathcal{O}(N_b N_p)$ and (v) $\mathcal{O}(N_g)$.

Table II. Computational complexity and scaling of the RMM-DIIS algorithm in terms of extensive system parameters. The scaling column is with respect to the $B \times G$ process grid. The ScaLAPACK process grids are typically chosen to be $\mathcal{O}(B^2)$ in size.

Description of Operation	Computational Complexity	Computational Scaling
$\tilde{v}_{\text{coul}}(\mathbf{r}), \tilde{v}_{\text{xc}}(\mathbf{r}), \tilde{v}(\mathbf{r})$	$\mathcal{O}(N_g)$	$\mathcal{O}(N_g/G)$
density mixing	$\mathcal{O}(N_g)$	$\mathcal{O}(N_g/G)$
(1st term in Eq. 3) $\times\tilde{\psi}_{ng}$	$\mathcal{O}(N_b N_g)$	$\mathcal{O}(N_b N_g/(BG))$
(2nd term in Eq. 3) $\times\tilde{\psi}_{ng}$	$\mathcal{O}(N_b N_g)$	$\mathcal{O}(N_b N_g/(BG))$
(3rd term in Eq. 3) $\times\tilde{\psi}_{ng}$	$\mathcal{O}(N_b N_p)$	
constructing $\tilde{n}(\mathbf{r})$	$\mathcal{O}(N_b N_g)$	$\mathcal{O}(N_b N_g/(BG))$
subspace diagonalization: ϵ_n, W_{mn}	$\mathcal{O}(N_b^3)$	$\mathcal{O}(N_b^3/B^2)$
subspace Cholesky decomposition including inversion: L^{-1}	$\mathcal{O}(N_b^3)$	$\mathcal{O}(N_b^3/B^2)$
constructing H_{mn}, S_{mn}	$\mathcal{O}(N_b^2 N_g)$	$\mathcal{O}(N_b^2 N_g/(BG))$
rotation of wave functions	$\mathcal{O}(N_b^2 N_g)$	$\mathcal{O}(N_b^2 N_g/(BG))$

A parallelization strategy employing only domain-decomposition (parallelization along the G-axis), leads to Amdahl's Law bottlenecks arising from the $\mathcal{O}(N_b^3)$ terms. GPAW is able to calculate these terms either using LAPACK [36], in which case the computation is replicated across all MPI tasks, or ScaLAPACK [37], in which case these terms are computed in a parallel fashion using a small subset of MPI tasks since $N_b \ll N_g$. Simultaneous parallelization on domains and bands (parallelization along the B- and G-axis) leads to an additional Amdahl's Law bottleneck arising from the $\mathcal{O}(N_g)$ terms whose computation is replicated along the B-axis.

Although it is not an Amdahl's Law bottleneck, the evaluation of the PAW projector terms leads to computational load imbalance in calculations where atoms are not evenly distributed geometrically. The computational complexity of evaluating the PAW projectors in real space is $\mathcal{O}(N_b N_p)$, much lower than in reciprocal space, which is $\mathcal{O}(N_b N_p N_g)$. This greatly reduces the total number of floating-point operations, especially for larger system sizes. However, the distribution of PAW

DOI: 10.1002/cpe

projectors (and their associated computational work) is determined by the domain-decomposition parallelization scheme. The total real-space grid is partitioned into equal-sized subdomains that may or may not contain projectors.[§] The effects are most readily observed for slab calculations with a vacuum, or any other configuration of atoms that leads to a non-uniform distribution of projectors, and thus computational load imbalance, both in terms of memory and in terms of floating-point operations. It is worth noting that this load imbalance could potentially be reduced by leveraging a computational runtime environment that can handle both global task scheduling (e.g., MADNESS [38]) and global shared-memory arrays (e.g., Global Arrays [39]), but this is currently an open issue for GPAW.

In spin-polarized cases and in periodic systems with k-points, Equation 1 can be solved independently for each spin and k-point. The computational complexity of most of the operations increases by a factor of $N_s N_k$ where N_s is the number of spins and N_k is the number of k-points. GPAW can parallelize over both the spin and k-point degrees of freedom, and because communication is needed only when constructing the charge density, the parallelization is very efficient. However, because the number of spins is limited to two, and the number of k-points is roughly inversely proportional to the number of atoms, the spin and k-point parallelization has only limited use in very large-scale calculations.

3.3. Communication Patterns

The GPAW code has four different types of communication patterns:

1. *Halo-exchange* communication patterns arising from FD stencil operations on $\tilde{n}$, $\tilde{v}_{\text{coul}}$, and $\tilde{\psi}_{ng}$ where boundary points have to be transferred between processes.
2. Projector-wave function integrals $P^a_{ni} = \sum_{g^a} \tilde{p}^a_{ig^a} \tilde{\psi}_{ng^a}$ where g^a are defined inside the augmentation sphere of atom a.
3. Parallel matrix multiplies for the construction of the H_{mn}, S_{mn}, and rotation operations on $\tilde{\psi}_{ng}$.
4. Dense linear algebra operations on $N_b \times N_b$ matrices, which are handled by ScaLAPACK.

Because the augmentation spheres extend typically only to adjacent domains, their communication pattern is three-dimensional nearest neighbor, similar to the halo-exchange.

Because of the specialized nature of the parallel matrix multiplies in GPAW, we chose to write our own routines on top of MPI and BLAS libraries. The routines themselves are complicated due to the need to support slightly irregular distributions of $\tilde{\psi}_{ng}$, as well as the PAW terms. The calculation of the matrix elements H_{mn} and S_{mn} requires communication between all the MPI tasks within band_comm. However, by utilizing a pipeline with overlapping communication and computation, the communication can in practice be performed as one-dimensional nearest-neighbor exchange. A more detailed description of our parallel matrix multiply algorithms is presented in Appendix B. Here we summarize the main features of our implementation.

1. Two distinct phases. A one-dimensional systolic ring algorithm implemented using MPI_Isend and MPI_Irecv on band_comm, immediately followed by MPI_Reduce on domain_comm. The MPI communicators band_comm and domain_comm are shown in Figure 1.
2. Internal blocking inside the ring algorithm to reduce memory usage and MPI message size. The amount of memory used for blocking is specified with the *buffer_size* parameter in units of kilobytes (KiB).
3. Use of ScaLAPACK for $\mathcal{O}(N_b^3)$ operations. Our present implementation uses a single two-dimensional process grid for all ScaLAPACK operations in a given real-space DFT calculation, but a different two-dimensional process grid is allowed for the LCAO initialization.

[§]GPAW allows N_g mod $G \neq 0$ which can lead to subdomains that are slightly different in volume; subdomains with significantly different volumes are not supported.

DOI: 10.1002/cpe

The algorithm for the rotation of wave functions by the unitary matrix W is similar except that MPI_Reduce is not needed and W is partially replicated, as noted in Section 3.1. Our parallel matrix multiply algorithms are scalable in terms of communication and memory, but a disadvantage is that it does not make use of as many networks links as would be possible if the MPI_Isend and MPI_Irecv were replaced by MPI collectives, particularly MPI_Bcast.

An important point in optimizing the communication pattern of GPAW is recognizing that communication patterns are radically different in terms of the amount of communication per MPI task. For example, the halo-exchange communication is proportional to the *surface area* of the subdomain $(N_g/G)^{2/3}$, while parallel matrix multiplies are proportional to the *volume* of the subdomain N_g/G. On the other hand, the projector-wave function integrals are independent of the domain size.

In principle, a four (or higher) dimensional torus network has a topology that can accommodate nearest neighbor communications for all these distinct communication patterns simultaneously. For example, the Blue Gene/P node can operate in virtual node (VN) mode, with each of the four cores in a node running an MPI task, effectively adding a very short fourth dimension to the three-dimensional torus network topology. We show in the results section that GPAW's scalability is primarily determined by optimizing the mapping so that the communication in the parallel matrix multiplies are nearest neighbors, since this is the worst communication pattern in the RMM-DIIS algorithm.

3.4. Initialization Phase, Single-core Peak Performance and Check Pointing

The initialization phase is defined to occur at the beginning of the calculation and to occur only once per calculation. It includes the LCAO initialization which itself is an entirely separate DFT algorithm that relies on direct diagonalization instead of iterative diagonalization. The GPAW LCAO code is described in greater detail in Larsen *et al.* [13]. Although the LCAO code has not been optimized as thoroughly as the real-space code, it contains sufficient parallelism to generate initial PS wave functions for a DFT calculation with at least $N_b \sim 10^4$.

Single-core peak performance is a measure of the efficiency with which software can use the floating point units on the computer cores. This efficiency is heavily dependent on the nature of algorithmic kernel and hardware details. Today's computer architectures use deep hierarchical levels of cache memory and have relatively slow access speeds to main memory. In the strong-scaling limit, the arrays used in a DFT calculation will become smaller for a fixed problem size. Since smaller arrays lead to a higher ratio of load operations to float operations, application developers tuning their computational kernels must work increasingly harder to retain reasonable percentages of the single-core peak performance.

For instance, vendor-supplied BLAS [40] libraries usually have double-precision general matrix multiply (DGEMM) function tuned to obtain $>75\%$ of single-core peak-performance on large square arrays. However, the arrays used in the GPAW's DGEMM calls include long skinny as well as small square arrays. The results obtained here used the optimized DGEMM available in the single-threaded version of IBM's ESSL [41], which has been primarily optimized for large square matrices where it obtains $>80\%$ of the single-core peak performance. In sharp contrast, DGEMM as used in GPAW typically only obtains $40\%-60\%$ of the single-core peak performance using ESSL. Furthermore, as a DFT calculation is strong scaled to larger numbers of computer cores, the dimensions of these arrays become even smaller, which reduces the single-core peak performance and increases the total wall-clock time of a calculation.

4. RESULTS

4.1. Preliminaries: Test Cases, Timers, TAU, and Blue Gene/P architecture

The performance characteristics of DFT codes are perhaps unique in the computational science community, in that there is no single algorithmic kernel that dominates SCF algorithms. Their distinguishing traits can be understood by studying the weak- and strong-scaling efficiency as a

DOI: 10.1002/cpe

function of system size parameters (N_a, N_b, N_g, N_p). In this section, we obtain performance data on several test cases with the GPAW code to gain insight into:

- The computational kernels that dominate the wall-clock time.
- The computational kernels that limit scalability.

This information allows us to construct a phenomenological map of GPAW's performance characteristics. Such a phenomenological map is valuable not only to DFT users, but also to DFT developers who are seeking to port their codes to new architectures and obtain good performance.

We study the performance of GPAW by collecting timing data for DFT calculations of bulk (face-centered cubic) gold for a variety of repeated cubic unit cells ($L_1 \times L_2 \times L_3$) listed in Table III. The cell sizes were chosen to facilitate comparison, while bulk gold was chosen for no other reason than to distinguish it from the proverbial test case in the condensed matter community (i.e., bulk silicon). As the length of the cell is doubled in a dimension, the total number of atoms, bands, and grid points is also doubled.

ScaLAPACK process grids were chosen to be square in shape and were not exhaustively tuned for performance. Our current implementation constrains all ScaLAPACK operations within a given calculation to a single process grid. Ideal weak-scaling with a fixed time per SCF cycle requires rescaling a two-dimensional process grid by a ratio proportional to N_b^3. However, rescaling a two-dimensional process grid by a ratio proportional to $\sqrt{(N_b^3)}$ is not generally possible, so we chose to rescale the ScaLAPACK process grid by a ratio proportional to N_b^2 instead. While it has been shown that 2.5- and 3-dimensional parallel dense linear algebra algorithms [42, 43] are required to achieve ideal scaling, we still expect that the two-dimensional algorithms that are used by ScaLAPACK will lead only to a linear increase in the time-to-solution as N_b increases; we show later in this section that this is not the case.

Table III. Extensive system and ScaLAPACK process grid parameters for bulk gold test cases.

L_1	L_2	L_3	N_a	N_b	N_g	ScaLAPACK process grid
2	4	4	128	712	(64, 128, 128)	2×2
4	4	4	256	1424	(128, 128, 128)	4×4
4	4	8	512	2848	(128, 128, 256)	8×8
4	8	8	1024	5696	(128, 256, 256)	16×16

All calculations used a grid spacing of $h = 0.125$Å and a gold PAW setup containing 18 PAW projector functions. Setups version 0.8.7929 was used which can be obtained online.[¶] Ground-state DFT calculations were performed in the local density approximation (LDA) [2] for 10 SCF iterations with a Γ-point sampling of the Brillouin zone. The template for our input file is included in Appendix A.

The performance data presented here was collected with the Tuning and Analysis (TAU) Performance System® [44, 45, 46, 47] (henceforth referred to simply as TAU). It consists of a number of tools for instrumenting code as well as storing and analyzing performance data. We have collected performance data by swapping out GPAW's default Python timers with those from TAU (shown as BGP_TIMERS in our figures). The performance data shown will be based on timers that are normally defined in a ground state DFT calculation (excluding those from the LCAO initialization) and are shown in Table IV. A `special font` is used in the main text to identify a timer. The timers represent exclusive timings unless noted otherwise. A number of timers that take up a negligibly small fraction of the total time have been intentionally omitted to make the figures easier to read.

The Blue Gene/P architecture has three networks for node-to-node communication [18]: (i) a global collective network, (ii) a global barrier network and (iii) a three-dimensional torus

[¶] `https://wiki.fysik.dtu.dk/gpaw/setups/setups.html`

DOI: 10.1002/cpe

Table IV. Timers in the SCF cycle as obtained from the output of the GPAW code. Right (left) indentation indicates child (parent) relationship in the hierarchy. Timers are non-overlapping at the same level. The sum of child timers does not necessarily provide complete coverage of the parent timer. Timers marked with an asterisk have been absorbed into their parent timer.

```
SCF-cycle
  Density
    Atomic density matrices*
    Mix*
    Multipole moments*
    Pseudo density*
      Symmetrize density*
  Hamiltonian
    Atomic*
      XC Correction
    Communicate energies
    Hartree integrate/restrict*
    Poisson
    XC 3D grid*
    vbar*
  Orthonormalize*
    Blacs Band Layouts*
      Inverse Cholesky
    calc_s_matrix
    projections
    rotate_psi
  RMM-DIIS
    Apply hamiltonian
    precondition
    projections
  Subspace diag*
    Blacs Band Layouts*
      Diagonalize
      Distribute results
    calc_h_matrix
      Apply hamiltonian
    rotate_psi
```

network. The global collective and global barrier networks handle MPI collectives and barrier calls, respectively, on the MPI_COMM_WORLD communicator, while the three-dimensional torus network handles all other types of MPI calls. The three-dimensional torus network is a low-latency high-bandwidth network where each node is directly connected to its six nearest neighbors with bi-directional links. A unique feature of the the Blue Gene/P architecture is that it supports partitions which are contiguous and allow different mappings of MPI tasks [48]. On a Blue Gene/P partition, a user's calculation is isolated and hence protected from external network traffic. A mapping refers to the assignment of MPI tasks on the three-dimensional torus network and turns out to be an important aspect of strong-scaling a GPAW calculation. For the work presented here, we use an MPI-only version of the GPAW code. Thus, all our results were obtained in VN mode where one MPI task runs on each of the four cores of the Blue Gene/P node.

DOI: 10.1002/cpe

4.2. Domain Decomposition and Band Parallelization

We begin by showing two performance enhancements implemented in the RMM-DIIS algorithm: (i) application of the KS operator to multiple wave functions as specified by the *blocksize* parameter and (ii) band parallelization (described in Section 3.2). Figure 2 shows the effect of these algorithmic strategies in isolation, as well as in tandem, for the smallest of our bulk gold test cases (2×4×4). The baseline case, corresponding to *blocksize* $= 1$ and $B = 1$, shows rapid performance degradation above 256 MPI tasks with $\{G_i\} = (4, 8, 8)$. The strong-scaling efficiency of the baseline case is significantly improved even for the $B = 1$ case by simply increasing to *blocksize* $= 10$. This improvement comes from a reduction in communication latency due to the consolidation of small MPI messages into larger ones. The total number of send/receives per MPI task is proportional to the total the number of $\hat{H}\tilde{\psi}_{ng}$ operations. Thus, applying the KS operator to multiple wave functions at a time, instead of one wave function at a time, reduces time spent in communication.[‖] Hence, performance data collected in later sections were obtained with *blocksize* $= 10$.

The efficacy of a domain decomposition only approach cannot continue indefinitely, because it is well known that these types of algorithms are limited by a surface-to-volume effect [49]. In Figure 2, we observe the strong-scaling efficiency falling below 75% at 1024 MPI tasks with $\{G_i\} = (8, 8, 16)$ for *blocksize* $= 10$ and $B = 1$. This behavior is in line with observations on other computer architectures, and we generally consider $N_{g_i}/G_i \sim 10$ for $i \in 1, 2, 3$ to be the minimum efficient size for a real-space domain on an MPI task. The best strong-scaling performance is achieved by simultaneously parallezing over bands and domains along with *blocksize* $= 10$, proving that our approach is tenable even for the smallest of our test cases.

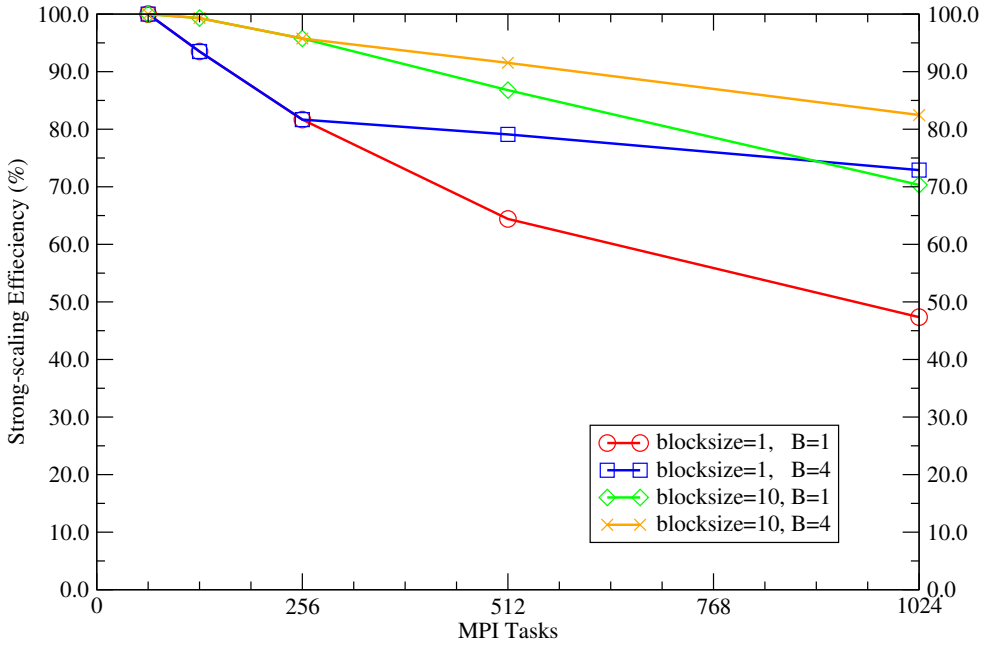


Figure 2. Strong-scaling efficiency as obtained by inclusive `SCF-cycle` timer for 10 SCF iterations of a ground-state DFT calculation on bulk gold 2×4×4 as a function of the *blocksize* and B. Relevant DFT calculations parameters are listed Table III.

We examine the *blocksize* $= 10$ and $B = 1$ case for bulk gold 2×4×4 in more detail to understand the underlying performance degradation. Figure 3a shows a breakdown of the inclusive time in `SCF-cycle` as a function of MPI tasks. This type of plot can be obtained from TAU's PerfExplorer

[‖]In our current implementation, only the communication from the projector-wave function integral is aggregated. The communication from the FD stencil operations on the wave function still occurs one band at a time.

DOI: 10.1002/cpe

and is called a fractional stacked-bar chart. The colors in the data bar of the stacked-bar chart are organized from left to right in the same order as the legend. Figure 3a shows that the strong-scaling bottlenecks at 1024 MPI tasks are `Communicate energies`, `Distribute results`, and `precondition`. We expect `precondition` to be among the computational kernels that are affected by a surface-to-volume effect, especially since these operations occur on a grid coarser than the wave function grid ($2h$ instead of h).

The presence of `Communicate energies` and `Distribute results` as bottlenecks is at first glance perplexing, but can be understood by looking at where these timers are located in the GPAW code. `Distribute results` and `Communicate energies` contain MPI collectives that are called after `XC Correction` and `Diagonalize`, respectively. Thus time spent in these timers is indicative of load imbalance in the *precedent* timers. While there are a number of ways to show this load imbalance and the relationship between these timers, it is most elegantly shown by a normal probability plot [50] obtained by PerfExplorer.

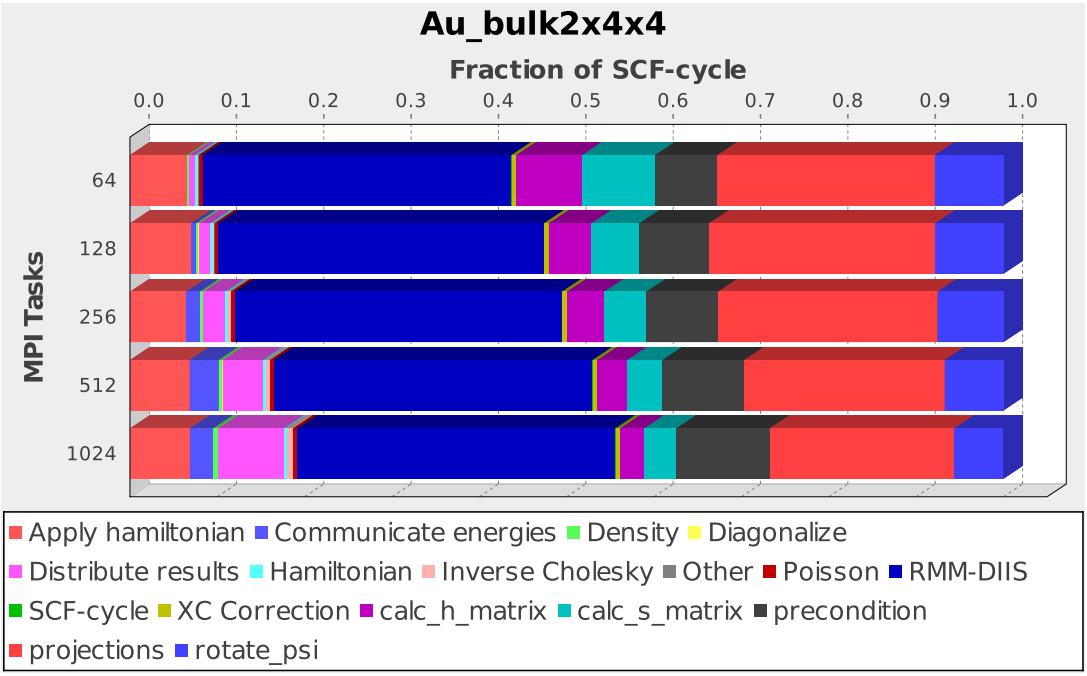
A normal probability plot is a graphical technique for assessing whether data is normally distributed. Data that is normally distributed falls on the ideal normal line, while data that departs from a normal distribution is shown by departures from the ideal normal line. Figure 3b shows that `XC Correction`, `Communicate energies`, `Diagonalize`, and `Distribute results` contain MPI tasks with timings outside statistical normality; this is in sharp contrast to `precondition`, whose data points falls on the ideal normal line. The load imbalance in `Diagonalize` arises because the dense diagonalization is performed on a subset of cores which form the 2×2 process grid. The load imbalance in `XC Correction` arises from exchange–correlation PAW corrections which are evaluated around each atom. In our current implementation, this computational work is local to domains containing atoms; thus load imbalance is present due to idle processes when the number of MPI tasks (=$B\times G$) is greater than N_a. Note that this computation is also replicated across band groups. In subsequent figures, we absorb `Communicate energies` into `XC Correction` and `Distribute results` into `Diagonalize` for the sake of clarity and simplicity.

4.3. Mapping and DCMF environment variables

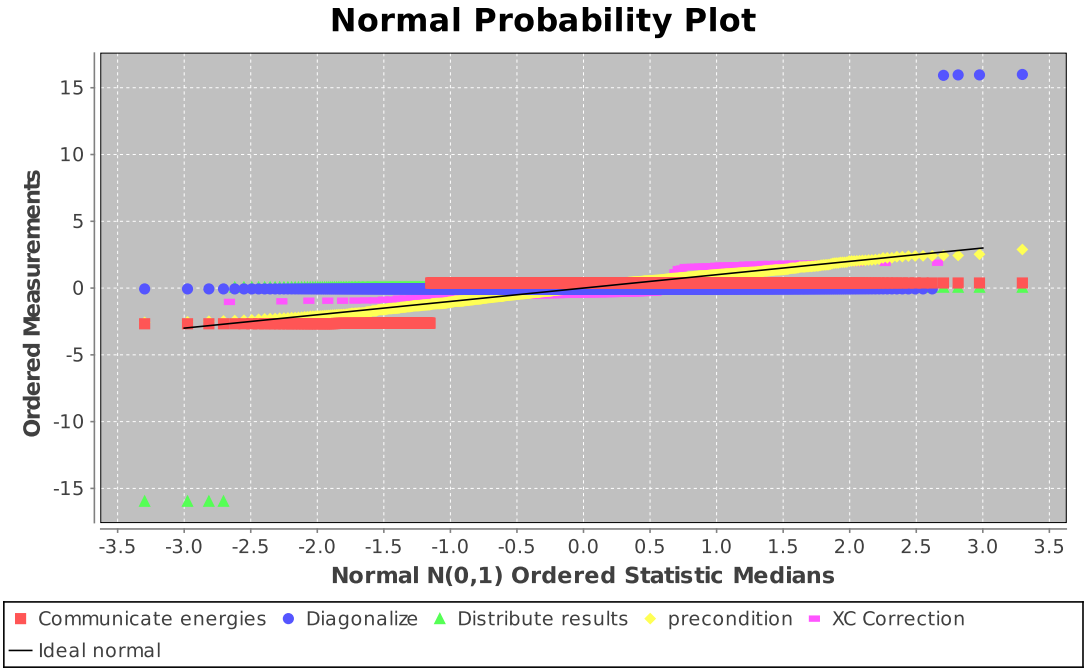
In order to minimize time-to-solution, we gathered timings for bulk gold $4\times4\times8$ at 2048 nodes while varying the values of the mapping, *buffer_size* parameter, and Deep Computing Messaging Framework (DCMF) [51] protocol variables. The standard mappings supported are permutations of XYZT, with T located either at the beginning or end of the mapping specification string. T can be thought of the dimension of parallelism available within the node and is equal to the number of MPI tasks per node ($T = 4$ for VN mode). While it is possible to specify a nonstandard mapping using a mapfile, it is most convenient for typical users to work with the standard mappings instead. By design, the MPI tasks in the GPAW's world communicator are ordered so that the band index is incremented last. The ordering of MPI tasks in GPAW's world communicator are fixed and not re-ordered as a function of the partition dimension. We have found that this approach is sufficient for scaling calculations that use simultaneously parallelization on bands and domains, but has shortcomings when parallelization is invoked on three or more layers of parallelization. For calculations which have simultaneous parallelization on bands and domains in conjunction with spins, k-points, and/or images, mapping should be accomplished by using a mapfile.

The 2048-node partitions on the Argonne Leadership Computing Facility (ALCF) Blue Gene/P have dimensions $\mathrm{XYZT} = 8\times8\times32\times4$ and can execute up to 8192 MPI tasks. For our computational experiments, we choose simultaneous parallelization on domains and bands with $\{G_i\} = (8, 8, 16)$ and $B = 8$ which uses all the cores on the 2048-node partition. Table V shows that for all cases, the MPI eager protocol yields the shortest time; this is because our parallel matrix multiply algorithm is able to overlap some communication and computation to the extent that is allowable by the hardware. It was discovered, somewhat counter-intuitively, that there is performance degradation when increasing *buffer_size* from 2048 KiB to 4096 KiB. We were able to confirm for select cases that this was due to the default value of DCMF_RECFIFO_SIZE, which is 8192 KiB per node. Because there are four MPI tasks per node in VN mode, four times the parallel

DOI: 10.1002/cpe



(a) Wall-clock time breakdown



(b) Normal probability plot at 1024 MPI tasks.

Figure 3. Ten SCF iterations of a ground-state DFT calculation on bulk gold $2\times4\times4$ with *blocksize* $= 10$ and $B = 1$. A complete set of timers and relevant DFT calculation parameters are listed Tables III and IV, respectively.

matrix multiply *buffer_size* must be a lower bound for the value of DCMF_RECFIFO – thus a *buffer_size* of 2048 KiB can already fill up the default DCMF_RECFIFO buffer. It would be possible

DOI: 10.1002/cpe

to benefit from a larger *buffer_size* value, but only at the expense of a larger DCMF_RECFIFO buffer value which would reduce the total amount of allocatable memory to GPAW.

Table V. Inclusive timings for SCF cycle (10 SCF iterations excluding LCAO initialization) of the bulk gold $4\times4\times8$ case at 2048 nodes as a function of mapping, parallel matrix multiply *buffer_size*, and DCMF protocol. The ScaLAPACK process grid was incidentally set to 6×6 instead of the value noted in Table III. The three DCMF protocols are eager, rendezvous (rzv) and optimized rendezvous (optrzv) [48]. Data was collected in virtual node mode (four MPI tasks per node) with DCMF_EAGER set to 8192 KiB. Mapping where the last index is equal to $B = 8$ are indicated with an asterisk (*).

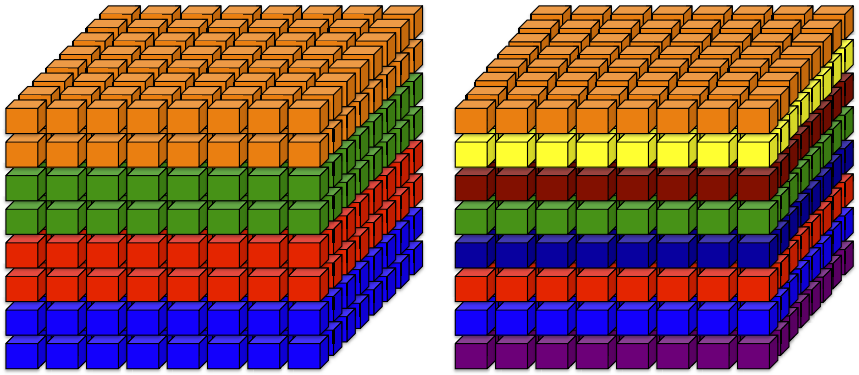
Mapping	Permuted Partition Dimensions	*buffer_size* (kilobytes)	Timing (seconds)		
			eager	rzv	optrzv
XYZT	$8\times8\times32\times4$	2048	592	619	610
		4096	612	628	629
TXYZ	$4\times8\times8\times32$	2048	471	532	520
		4096	500	532	527
TZYX*	$4\times32\times8\times8$	2048	462	479	478
		4096	459	532	628

Lastly, we see that the largest performance enhancement comes from an optimal mapping of MPI tasks unto the torus network. This optimal mapping can be achieved when B is identical to the last index of the Blue Gene/P mapping. While in principle an optimal mapping could always be obtained on a four-dimensional network, architectural constraints prevent the use of certain values of $\{G_i\}$ and B. The most obvious constraint is the total number of available cores per node (T dimension) and the available set of partition shapes and sizes (X, Y, and Z dimensions) available on the Blue Gene/P at ALCF. We illustrate this concept using a 512-node partition for simplicity in Figure 4. This figure depicts a 512-node partition with cartesian dimensions $8\times8\times8$. Figure 4a shows a decomposition of $B = 4$ along one of the partition dimensions. As discussed in Section 3.3, our parallel matrix multiply involves a large volume of communication in a one-dimensional systolic ring pattern. There are large messages sent using MPI_Isend and MPI_Irecv between each MPI process and its adjacent MPI process in band_comm. This communicator is not explictly shown, but is defined to be the set of MPI processes forming columns perpendiculator to the colored plane. In Figure 4a, band_comm consist of the perpendicular column of MPI processes from every other (i.e., second nearest neighbor) colored plane of nodes. On the other hand, Figure 4b shows a decomposition of $B = 8$, where band_comm consists of the perpendicular column of MPI process from adjacent (i.e., first nearest neighbor) colored planes of nodes. The one-dimensional systolic communication pattern in the $B = 4$ case leads to hopping over a plane of nodes and thus significant network contention. The $B = 8$ decomposition requires no hopping over plane of nodes and leads to much lower network contention.

4.4. Performance data for bulk gold

We collected strong-scaling performance data for our four bulk gold benchmark test cases. The parameters for each of the cases are given in Table III. For each case, we created a *stacked-bar* and *fractional stacked-bar* chart using TAU's PerfExplorer. The stacked-bar chart shows both the strong-scaling performance and the wall-clock time breakdown, while the fractional stacked-bar chart shows only the percentage wall-clock time breakdown. The fractional stacked-bar chart can be considered a magnified view of the stacked-bar chart view that more clearly depicts fine grain details. In the *stacked-bar* chart representation, ideal strong-scaling is shown by bars which are reduced proportionately between MPI tasks. For example, between 64 and 128 MPI tasks, the bars for each computational kernel would go down by a factor of 2 in the limit of ideal scalability. In the *fractional stacked-bar* chart representation, ideal strong-scaling performance is shown by a constant fraction of the wall-clock time with varying MPI tasks. These two representations for the performance data are complementary and helpful in interpreting our results. Unless noted otherwise,

DOI: 10.1002/cpe



(a) Bad mapping for $B = 4$ (b) Good mapping for $B = 8$

Figure 4. Mappings available on a 512-node partition of Blue Gene/P with cartesian dimensions equal to $8\times8\times8$. Cubes represent computer nodes. Nodes with the same color form planes of MPI processes that exist on the same domain_comm; band_comm is orthogonal to these planes and is not shown.

our results were obtained with *blocksize* set to 10, *buffer_size* set to 2048 KiB, DCMF_EAGER set to 8192 KiB, and B set to the last index of the Blue Gene/P mapping (whenever this was possible with the given partition).

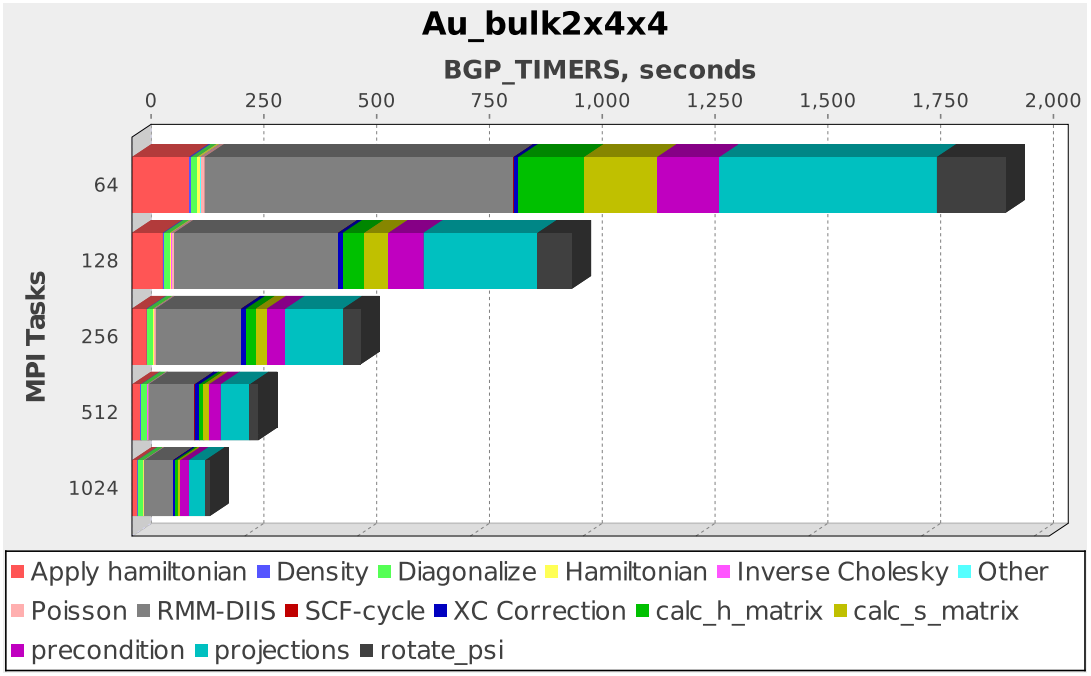
Our smallest test case is $2\times4\times4$ which was discussed in greater detail in Section 4.2. We only summarize the results here. Figure 5b shows that 74% of the inclusive `SCF-cycle` time is spent on $\hat{H}\tilde{\psi}_{ng}$ products (sum of `Apply hamiltonian`, `projections`, `precondition`, `RMM-DIIS`), which are either $\mathcal{O}(N_b N_g)$ or $\mathcal{O}(N_b N_p)$ operations, while only about 24% is spent in parallel matrix multiplications (sum of `calc_s_matrix`, `calc_h_matrix`, and `rotate_psi`), which are $\mathcal{O}(N_b^2 N_g)$ operations.** As the $2\times4\times4$ case is strong-scaled to a larger number of MPI tasks, the bottlenecks emerging are `XC correction` arising from load imbalance, `Diagonalize` arising from Amdahl's Law, and `precondition` arising from a surface-to-volume effect.

The dominant computational kernels in the $4\times4\times4$ case are very similar to those in the $2\times4\times4$. However, it is worth noting that strong-scaling bottlenecks are different for $4\times4\times4$ than for $2\times4\times4$. Figure 6b shows that `XC correction` and `Diagonalize` are still an issue, but that `precondition` is no longer as relevant even though $N_{g_i}/G_i \sim 10$ for $i \in 1, 2, 3$ per MPI task. In addition, there are a number of subtle bottlenecks that appear at 4096 MPI tasks ($B = 16$). For instance, `Poisson` and `Inverse Cholesky` no longer take a negligible fraction of the inclusive `SCF-cycle` time, which is consistent with Amdahl's Law. We identified these inherent limitations of our parallelization approach in Section 3.1 (see Table II); namely, that the `Poisson` computation is replicated for each group of bands and that `Inverse Cholesky` is an operation that is constrained to the concurrency available in the ScaLAPACK process grid which is only $4 \times 4 = 16$ MPI tasks for this test case.

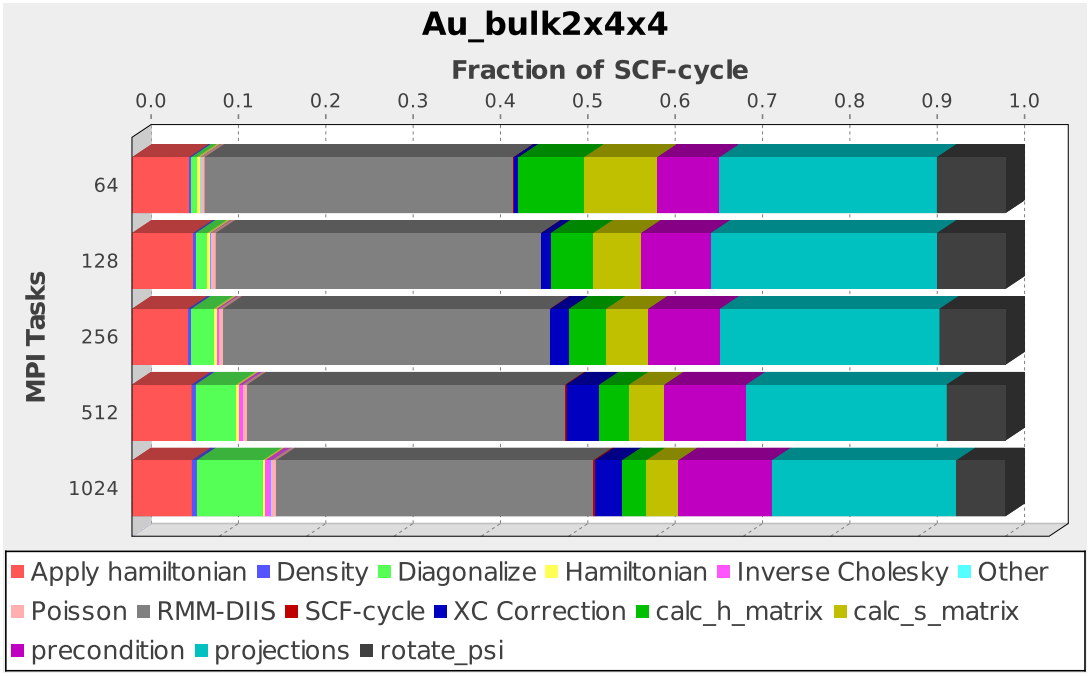
The $4\times4\times8$ case is the second largest test case we consider and is significantly different from the $2\times4\times4$ with respect to the dominant computational kernels. Figure 7b shows that 50% of the inclusive `SCF-cycle` time is spent in parallel matrix multiplies, while 46% is spent in $\hat{H}\tilde{\psi}_{ng}$ products. This test case is interesting because it marks a transition region where the dominant computational kernels shift from the $\hat{H}\tilde{\psi}_{ng}$, which are $\mathcal{O}(N^2)$ in computational complexity, to parallel matrix multiplies which are $\mathcal{O}(N_b^2 N_g)$. As the $4 \times 4 \times 8$ case is strong-scaled out, the largest bottlenecks are `XC Correction` and `Diagonalize`, but `Inverse Cholesky` is also more prominent.

**These percentages can be approximately obtained from visual inspection of the fraction stacked-bar chart.

DOI: 10.1002/cpe



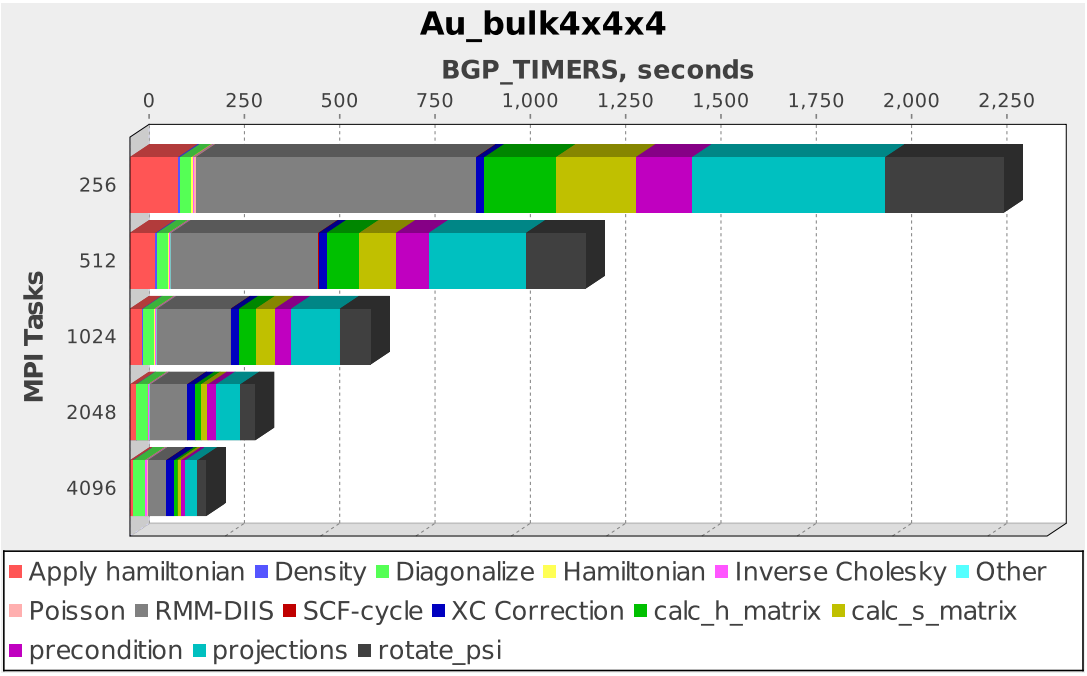
(a) Stacked-bar chart



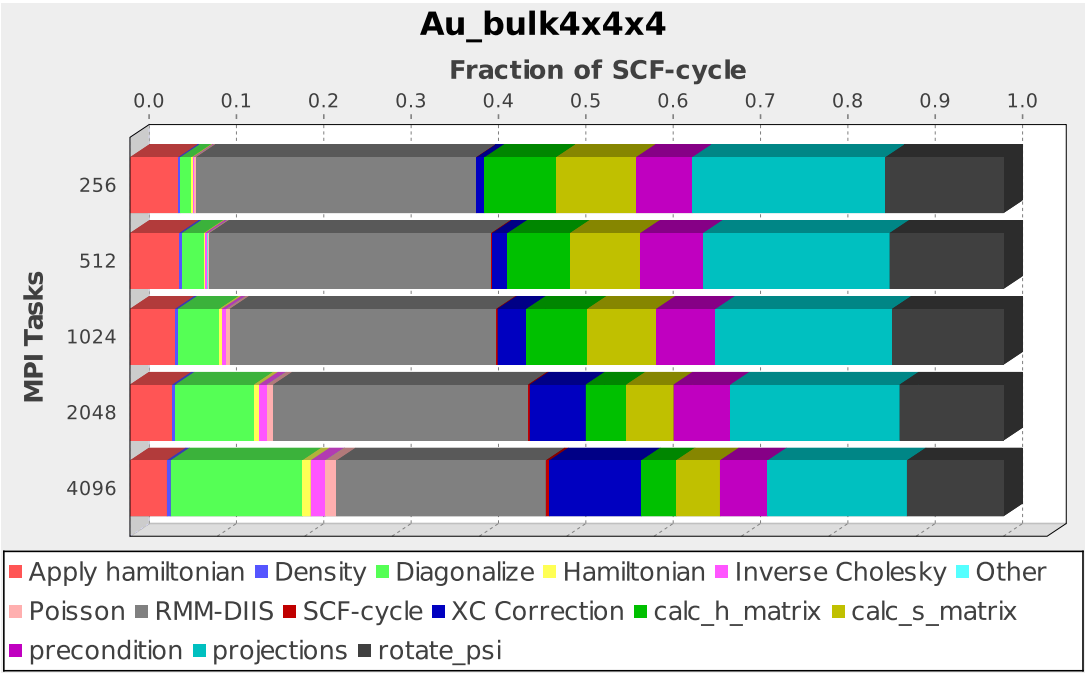
(b) Fractional stacked-bar chart

Figure 5. Strong-scaling performance data as obtained by timers for 10 SCF iterations of a ground-state DFT calculation on bulk gold $2\times4\times4$. A complete set of relevant DFT calculation parameters and timers are listed Tables III and IV, respectively.

DOI: 10.1002/cpe



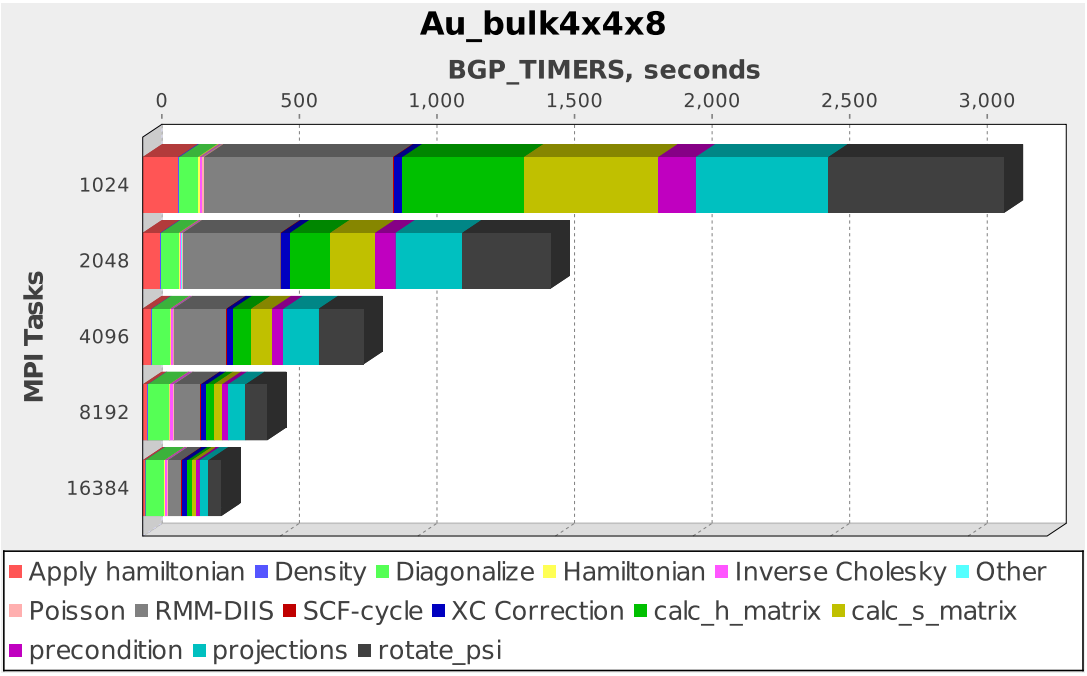
(a) Stacked-bar chart



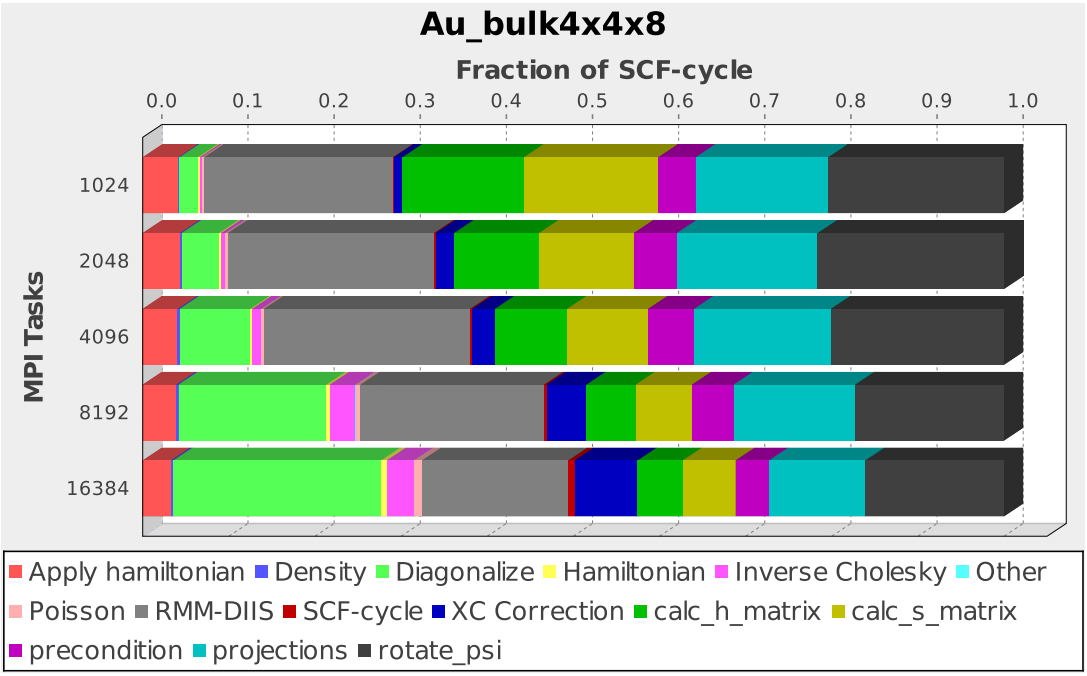
(b) Fractional stacked-bar chart

Figure 6. Strong-scaling performance data as obtained by timers for 10 SCF iterations of a ground-state DFT calculation on bulk gold $4\times4\times4$. A complete set of relevant DFT calculation parameters and timers are listed Tables III and IV, respectively.

DOI: 10.1002/cpe



(a) Stacked-bar chart



(b) Fractional stacked-bar chart

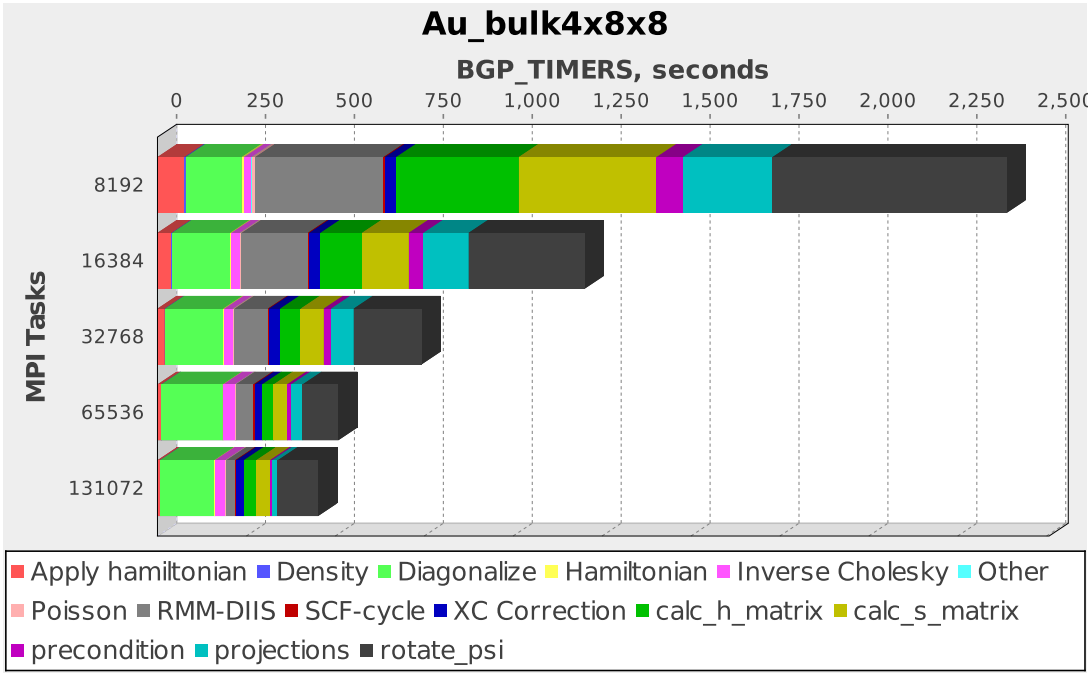
Figure 7. Strong-scaling performance data as obtained by timers for 10 SCF iterations of a ground-state DFT calculation on bulk gold $4\times4\times8$. A complete set of relevant DFT calculation parameters and timers are listed Tables III and IV, respectively.

DOI: 10.1002/cpe

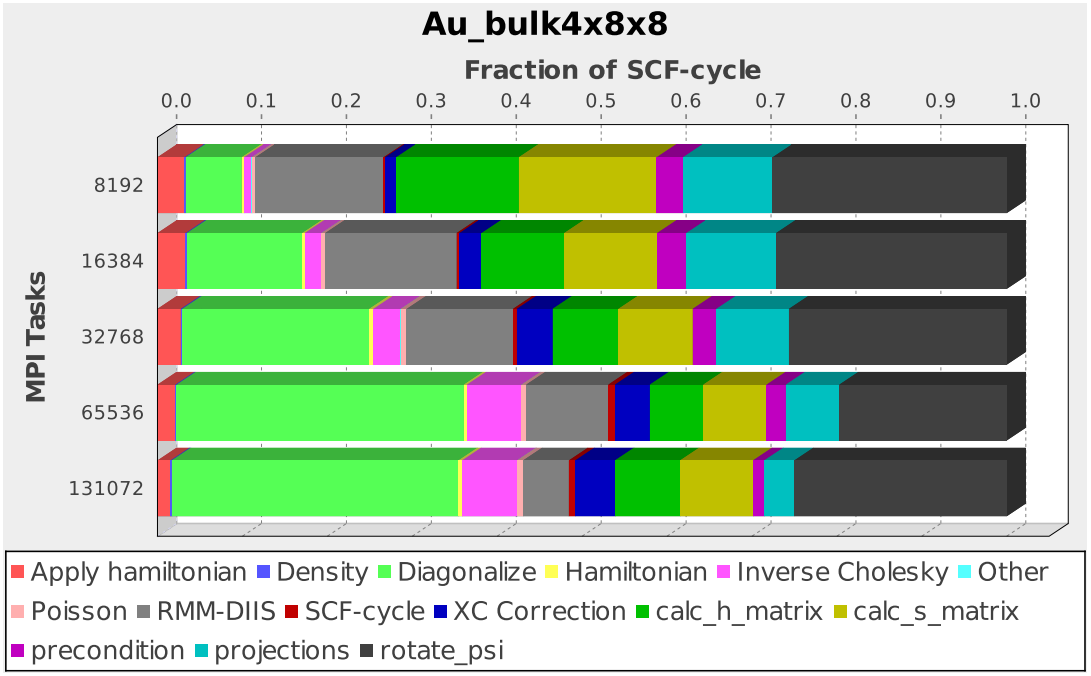
Our largest test case investigated in this study is the $4\times8\times8$ shown in Figure 8. We observe that 58% of the inclusive `SCF-cycle` time is spent in parallel matrix multiplies, while only 32% is spent in $\hat{H}\tilde{\psi}_{ng}$ products. As the $4\times8\times8$ test case is strong-scaled out, we see in Figure 8b that `Diagonalize` is the dominant computational kernel at 131,072 MPI tasks. In addition, at this very large scale, we discover that our parallel matrix multiplies are not scaling as well as in the previously discussed smaller test case ($4\times4\times8$). This is most easily seen in Figure 7a by comparing the exclusive time of `calc_h_matrix`, `calc_s_matrix`, and `rotate_psi` between 65,536 and 131,072 MPI tasks. We speculate that the poor scaling comes from the mapping not being commensurate with the value of B=64 at 131,072 MPI tasks. This hypothesis is based on our previous observations of timings for $4\times4\times8$ test case which exhibited a 28% performance penalty (compare timings for *buffer_size* $= 2048$ for eager between XYZT and TZYX in Table V). Currently, it is not possible to accommodate an ideal mapping for $B > 32$ with the available partitions on the Blue Gene/P at the ALCF.

It is evident from Figures 7–8 that `Diagonalize` is the predominant factor for the poor performance in the two largest test cases. The subspace diagonalization in GPAW is performed by ScaLAPACK's Divide and Conquer dense diagonalization routine (PDSYEVD) [52]. Its weak-scaling performance is shown in Figure 9, where we plot the ideal versus the observed wall-clock time. This behavior is consistent with the findings of Sunderland [53] and Petschow [54]. While we have not investigated the source of the poor performance of PDSYEVD on Blue Gene/P, we note that Petschow identifies the reduction to tridiagonal form as a severe bottleneck when the upper triangle of the input matrix is referenced – which is the current implementation GPAW. However, the performance issue note by Petschow et. al. becomes prevalent only at matrix sizes larger than 10,000, which is almost a factor of two larger than what is studied in this paper.

DOI: 10.1002/cpe



(a) Stacked-bar chart



(b) Fractional stacked-bar chart

Figure 8. Strong-scaling performance data as obtained by timers for 10 SCF iterations of a ground-state DFT calculation on bulk gold $4\times8\times8$. A complete set of relevant DFT calculation parameters and timers are listed Tables III and IV, respectively.

DOI: 10.1002/cpe

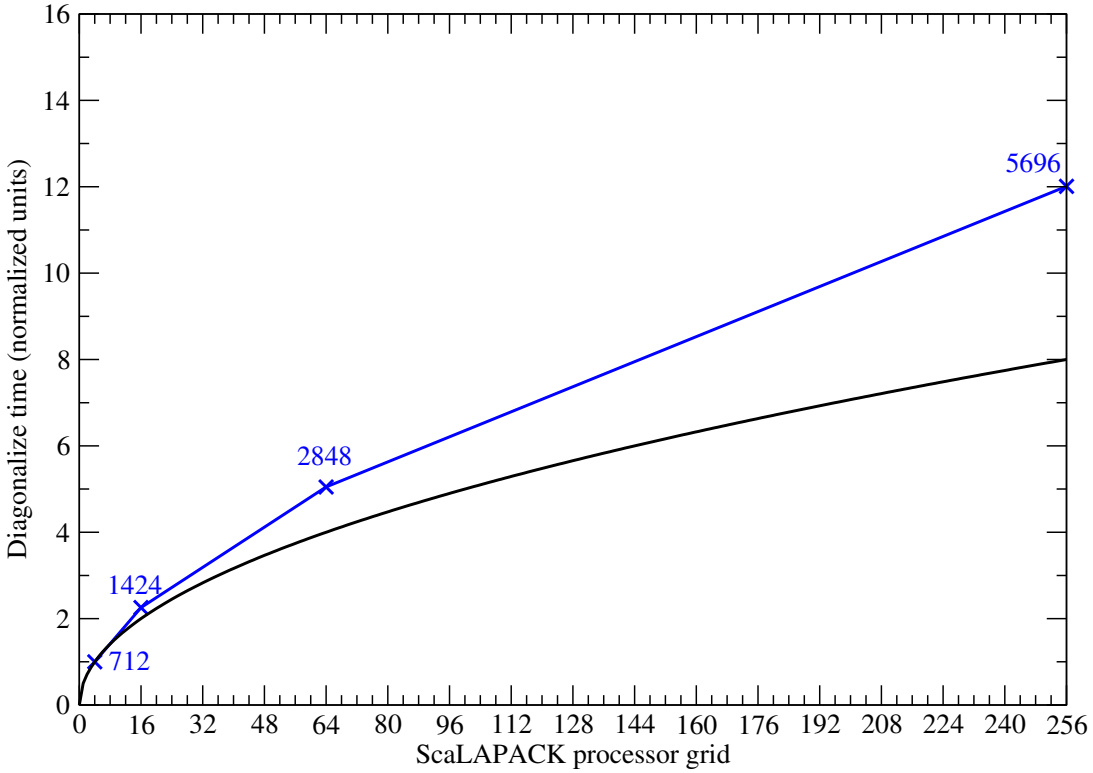


Figure 9. Weak-scaling performance of PDSYEVD for bulk gold test cases as a function of the ScaLAPACK process grid. The value of N_b is annotated adjacent to the data point (blue crosses). The data points are the `Diagonalize` time normalized to the value at $N_b = 712$. Worse performance is indicated by data points (blue crosses) above the ideal line (black).

5. CONCLUSIONS

The scientific need for DFT calculations on increasingly larger system sizes as well as the availability of massively parallel supercomputers has driven the development of more scalable algorithms. The GPAW data structures are completely scalable with respect to memory with the possible exception of $\mathcal{O}(N_a)$ arrays. Our real-space implementation of PAW method is parallelized over all quantum mechanical indices: k-points, spins, bands and domains. We have examined the weak and strong-scaling behavior of several bulk gold test case ranging in size from N_b=712 to N_b=5696.

We determined that proper mapping of MPI tasks on the Blue Gene/P torus is an important aspect of obtaining good performance. For $N_b{\sim}500{-}2000$, the dominant part of the inclusive `SCF-cycle` time was $\hat{H}\tilde{\psi}_{ng}$ products which contain all the $\mathcal{O}(N^2)$ terms. For $N_b{>}2000$, the dominant part of the `SCF-cycle` are parallel matrix multiplies with computational complexity $\mathcal{O}(N_b^2 N_g)$. On all test cases investigated, `Diagonalize` was an Amdahl's Law bottleneck, consistent with findings of Kent [27]. As N_b increases, terms with $\mathcal{O}(N_b^3)$ computational complexity become an increasingly severe bottleneck for weak- and strong-scaling. A less severe, but non-neglible, strong-scaling bottleneck included `XC correction`, which suffers from load imbalance as well as computation replicated across the different band groups (B-axis). For the largest calculation presented in this work, our parallel matrix multiply algorithm exhibited poor performance at 131,072 MPI tasks, which was caused by the inability to obtain an optimal mapping on the Blue Gene/P partition. A possible way to improve upon our current parallel matrix multiply algorithm is to replace our MPI_Isend/MPI_Irecv calls with MPI_Bcast, which would make better use of the Blue Gene/P three-dimensional torus network.

DOI: 10.1002/cpe

We have shown that ground-state DFT calculations up to $N_a{\sim}1000$ with $N_b{>}5000$ are feasible with massively parallel supercomputers such as Blue Gene/P. Similarly large calculations have been around since at least 2006, where F. Gygi et. al. [55] performed similarly large calculations on Blue Gene/L which result in resulted in a Gordon Bell Award. The major differences being the use of uniform real-space grids instead of a plane-wave basis and the use of PAW instead of norm-conserving pseudopotentials. Kleis *et al.* [56] performed total energy calculations on gold nanoparticles containing as many as 1415 atoms ($N_b > 8064$) using GPAW on ALCF's Blue Gene/P. More recently, Lin *et. al* [57] performed similarly large scale DFT calculations on platinum nanoparticles. While the work presented here focuses on the Blue Gene/P architecture, many architectures have benefited from the parallelization strategy presented here; most notably, Cray XT5 (see Figure 14 in Reference [5]), and even commodity Linux clusters. The size of the largest test case investigated in this work is near the practical limits accessible by $\mathcal{O}(N^3)$ DFT methods. While it is possible to push this limit further with diagonalization-free approaches in conjunction with more scalable parallel dense linear algebra algorithms, robust $\mathcal{O}(N)$ DFT methods are needed to make calculations on $N_b{\sim}10^4$ practical for time-sensitive calculations such as *ab initio* molecular dynamics.

When going beyond ground-state DFT calculations, new possibilities for utilizing massively parallel supercomputers arise. For example, time-dependent DFT (both in linear response and real-time propagation forms), GW-approximation, and Bethe-Salpeter equation in GPAW have been parallelized with additional degrees of freedom with minimal communication requirements. These are also computationally much more demanding, and are thus currently limited to much smaller system sizes than ground-state studies. The optimization work reported here also paves the way for enabling these beyond-DFT schemes in much larger parallel scale than previously possible.

ACKNOWLEDGMENTS

We thank Marcin Dułak from the Center for Atomic-scale Materials Design (CAMd) for the initial porting of GPAW to the Blue Gene/P at the Argonne Leadership Computing Facility (ALCF). We thank Nils Smeds (IBM Sweden) and William Scullin (ALCF) for useful discussions on Python-related issues on the Blue Gene/P architecture. We thank John Linford from ParaTools, Inc. and Kevin Huck from University of Oregon for assistance in generating the figures using TAU's PerfExplorer. We acknowledge Kalyan Kumaran (ALCF) and Jeff Greeley (Purdue University) for their support of this project. We are grateful to Carolyn M. Steele from Argonne National Laboratory's (ANL's) Communications, Education and Public Affairs (CEPA) division for reviewing and editing this manuscript. This work has been supported by the Academy of Finland (Project 110013 and the Center of Excellence program) and Tekes MASI-program. We acknowledge support from the Danish Center for Scientific Computing (DCSC). CAMd is sponsored by the Lundbeck Foundation. The research at the University of Oregon was supported by grants DOE ER26057, ER26167, ER26098, and ER26005 from the U.S. Department of Energy, Office of Science. This research used resources of the ALCF at ANL, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. A.H.L. acknowledges support from the European Research Council Advanced Grant DYNamo (ERC-2010-AdG Proposal No. 267374) and Grupo Consolidado UPV/EHU del Gobierno Vasco (IT578-13).

A. BENCHMARK INPUT FILE TEMPLATE

```
# Perform a single point total energy calculation
from ase import Atoms
from gpaw import GPAW, Mixer, ConvergenceError, PoissonSolver
from gpaw.poisson import PoissonSolver
from gpaw.utilities.timing import Timer, TAUTimer
from gpaw.eigensolvers.rmm_diis import RMM_DIIS
```


DOI: 10.1002/cpe

```
ps = PoissonSolver(nn='M', relax='GS', eps=1e-9)
es = RMM_DIIS(keep_htpsit=False, blocksize=10)

# dimensions
L1 = <value1>
L2 = <value2>
L3 = <value3>

# system
nbandspercell = 22.25
a = 4.08
bulk = Atoms('Au4',
             positions=((0, 0 ,0),
                        (0.5, 0.5, 0),
                        (0.5, 0, 0.5),
                        (0, 0.5, 0.5)),
             pbc=True)

bulk.set_cell((a, a, a), scale_atoms=True)
bulk = bulk.repeat((L1, L2, L3))

calc = GPAW(h=0.1275,
            maxiter=10,
            mode='fd',
            poissonsolver=ps,
            nbands=int(nbandspercell * L1 * L2 * L3),
            spinpol=False,
            xc='LDA',
            width=0.01,
            mixer=Mixer(0.10, 5, 100.0),
            eigensolver=es,
            parallel={'buffer_size':2048},
            txt='Au_bulk.out')

bulk.set_calculator(calc)

try:
    bulk.get_potential_energy()
except ConvergenceError:
    pass
```

B. PARALLEL MATRIX MULTIPLY

There are two different types of parallel matrix multiplication routines needed by the RMM-DIIS algorithm. Although both types have computational complexity of $\mathcal{O}(N_b^2 N_g)$, different routines are needed because of the distinct shapes and logical distributions for the input matrices. One routine is needed for the construction of H_{mn}, S_{mn}, and another one for performing rotation operations on $\tilde{\psi}_{ng}$. The former has the general form of $O_{mn} = \langle\tilde{\psi}_m|\hat{O}|\tilde{\psi}_n\rangle$, where $\hat{O}$ is a matrix-free operator. Let $\tilde{\phi}_{ng}$ denote the matrices resulting from $\hat{O}|\tilde{\psi}_n\rangle$. We denote the row and column coordinates of the MPI task in the two-dimensional process grid by (r, c). Additionally, let B_r and G_c denote the range of indices of $\tilde{\psi}_{ng}$ associated with MPI task (r, c).

DOI: 10.1002/cpe

We calculate O_{mn} in terms of an auxiliary matrix denoted $O^{1D}[q](r,c)$, where each entry q is a $N_b/B \times N_b/B$-size subblock of O_{mn}. The superscript reminds us that this matrix contains one-dimensional slices of the O_{mn} matrix. Since O_{mn} is symmetric, only $q = B$ div 2 exchanges are needed to build up all the needed subblocks to form the lower triangle of O_{mn}.[††] The capability to *only* compute half the matrix in parallel has substantial computational savings and is not currently available in any parallel dense linear algebra package.

Our parallel matrix multiply algorithm proceeds as follows:

1. Initially, without any communication, each MPI task can calculate its diagonal block contribution to O_{mn}, which is given by

$$O^{1D}[0](r,c) = \sum_{g}^{G_c} \tilde{\phi}^*_{mg} \tilde{\psi}_{mg} \Delta V, \quad \{m \in B_r\}, \tag{6}$$

   where ΔV is the volume element for the real-space grid. Then each MPI task in band_comm exchanges $\tilde{\phi}_{ng}$ with the nearest neighbor rank and computes the next contribution. In general, for the q-th wave function exchange, we have

$$O^{1D}[q](r,c) = \sum_{g}^{G_c} \tilde{\phi}^*_{mg} \tilde{\psi}_{ng} \Delta V, \quad \{m \in B_{(r+q) \operatorname{div} B}, n \in B_r\}. \tag{7}$$

   This phase of the communication uses MPI_Isend and MPI_Irecv. This allows us to overlap the computation phase of $O[q](r,c)$ with the communication of $O[q+1](r,c)$ for $q < B$ div 2. Instead of exchanging the entirety of $\tilde{\phi}_{ng}$ in a single pass, a *buffer_size* parameter is available to control the chunk of $\tilde{\psi}_{ng}$ that is sent and received (not explictly shown in the equations). While this additional layer of complexity is tedious to code, the additional memory savings are beneficial on low-memory nodes like those found on Blue Gene/P.
2. Summing along the columns of the two-dimensional process grid (MPI_Reduce on domain_comm), we have

$$O^{1D}[q](r, c=0) = \sum_{c} O^{1D}[q](r,c). \tag{8}$$

3. At this step, we have all the necessary blocks of O_{mn}, but they are not distributed in a form that is amenable to use with ScaLAPACK. This requires two additional steps.
   (a) Since the necessary subblocks are not on the needed task on the $B \times G$ process grid, an MPI-based routine assembles the blocks of $O^{1D}[q](r, c=0)$ into a one-dimensional *column-wise* block layout: $O^{1D}_{mn}(r, c=0) := O_{mn}(r, c=0), \quad \{\forall m, n \in B_r\}$
   (b) A ScaLAPACK routine call redistributes $O^{1D}_{mn}(r,c)$ to the two-dimensional block cyclic layout required by the ScaLAPACK library. The dimensions of the ScaLAPACK process grid are a user specified parameter and independent of the $B \times G$ process grid. After the ScaLAPACK-based operations are completed, a second call to a ScaLAPACK routine is required to return the resulting matrix to its previous one-dimensional *column-wise* block layout.

The second type of parallel matrix multiply has the general form of $\tilde{\psi}'_{mg} = \sum_n U_{mn} \tilde{\psi}_{ng}$ where U_{mn} is a unitary matrix. Our parallel matrix multiply algorithm is similar to the work of Solomonik [42] and proceeds as follows:

1. U_{mn} is computed by ScaLAPACK in its native two-dimensional block cyclic layout as the result of subspace diagonalization or the Cholesky factorization plus triangular inversion.

[††] In general, for odd values of B greater than unity, exactly the lower triangle of O_{mn} is computed. For even values of B greater than two, a few subblocks in addition to the lower triangle are computed. In the limit of large even B, this additional computation is negligible.

DOI: 10.1002/cpe

2. Using a ScaLAPACK routine, U_{mn} is redistributed from a two-dimensional block cyclic layout into a one-dimensional *row-wise* block layout that only exists on the root node of domain_comm: $U^{1D}_{mn}(r, c = 0) := U_{mn}(r, c = 0), \quad \{m \in B_r, \forall n\}$.
3. $U^{1D}_{mn}(r, c = 0)$ is broadcast along the columns of the two-dimensional process grid (MPI_Broadcast on domain_comm). We define a new array $U^{1D}[q](r)$ in a manner reminiscent of our previously described matrix multiply. The c index is dropped as the matrix is replicated on the G-axis and the q index is introduced as this matrix is accesssed in $N_b/B \times N_b/B$ subblocks.
4. Each MPI task calculates its local contribution to the rotated wave functions:

$$\tilde{\psi}'_{mg}(r, c) = \sum_{n}^{B_r} U^{1D}[0](r)\tilde{\psi}_{ng}(r, c), \quad \{m \in B_r, g \in G_c\}. \tag{9}$$

Since symmetry is not as easily exploitable here, $q = B - 1$ exchanges are required to rotate the wave functions. In general, the q-th exchange is given by

$$\tilde{\psi}'_{mg}(r, c) = \tilde{\psi}'_{mg}(r, c) + \sum_{n}^{B_r} U^{1D}[q](r)\tilde{\psi}_{ng}(r, c), \quad \{m \in B_r, g \in G_c\}, \tag{10}$$

where the $\tilde{\psi}'_{mg}$ on the right-hand side contains the cumulative contributions from the previous $q - 1$ exchanges. The communication pattern here closely follows the prior algorithm with respect to the overlapping computation and communication with MPI_Isend and MPI_Irecv. We also use the *buffer_size* parameter to control the chunk of $\tilde{\psi}_{ng}$ that is sent and received.

We schematically depict the the distinct MPI operations on the rows and columns of the process grid below:

Table VI. MPI operations in parallel matrix multiply routines

Operation	Before	After
ISend/IRecv	band_comm rank 0 | rank 1 | rank 2 $\psi_{B_0G_c}$ | $\psi_{B_1G_c}$ | $\psi_{B_2G_c}$	band_comm rank 0 | rank 1 | rank 2 $\psi_{B_2G_c}$ | $\psi_{B_0G_c}$ | $\psi_{B_1G_c}$
Reduce	domain_comm rank 0 | rank 1 | rank 2 $O[q](r,0)$ | $O[q](r,1)$ | $O[q](r,2)$	domain_comm rank 0 | rank 1 | rank 2 $\sum_c O[q](r,c)$ | |
Bcast	domain_comm rank 0 | rank 1 | rank 2 $U^{1D}_{mn}(r,0)$ | |	domain_comm rank 0 | rank 1 | rank 2 $U^{1D}_{mn}(r,0)$ | $U^{1D}_{mn}(r,0)$ | $U^{1D}_{mn}(r,0)$

REFERENCES

1. Hohenberg P, Kohn W. Inhomogeneous electron gas. *Phys. Rev.* 1964; **136**:B864–B871.
2. Kohn W, Sham LJ. Self-consistent equations including exchange and correlation effects. *Phys. Rev.* 1965; **140**:A1133–A1138.
3. Amdahl G. Validity of the single processor approach to achieving large-scale computing capabilities. *American Federation of Information Processing Societies Conference Proceedings* 1967; **30**:225–231.
4. Mortensen JJ, Hansen LB, Jacobsen KW. Real-space grid implementation of the projector augmented wave method. *Phys. Rev. B* Jan 2005; **71**:035 109, doi:10.1103/PhysRevB.71.035109.
5. Enkovaara J, Rostgaard C, Mortensen JJ, Chen J, Dułak M, Ferrighi L, Gavnholt J, Glinsvad C, Haikola V, Hansen HA, *et al.*. Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method. *J. Phys.: Condens. Matter* 2010; **22**:253 202.

DOI: 10.1002/cpe

6. Chelikowsky JR, Troullier N, Saad Y. Finite-difference-pseudopotential method: Electronic structure calculations without a basis. *Phys. Rev. Lett.* 1994; **72**:1240–1243.
7. Briggs EL, Sullivan DJ, Bernholc J. Large-scale electronic-structure calculations with multigrid acceleration. *Phys. Rev. B* 1995; **52**:R5471–R5474.
8. Blöchl PE. Projector augmented-wave method. *Phys. Rev. B* 1994; **50**:17 953–17 979.
9. Walter M, Häkkinen H, Lehtovaara L, Puska M, Enkovaara J, Rostgaard C, Mortensen JJ. Time-dependent density-functional theory in the projector augmented- wave method. *J. Chem. Phys.* 2008; **128**(24):244101, doi:10.1063/1.2943138. URL `http://link.aip.org/link/?JCP/128/244101/1`.
10. Yan J, Mortensen JJ, Jacobsen KW, Thygesen KS. Linear density response function in the projector augmented wave method: Applications to solids, surfaces, and interfaces. *Phys. Rev. B* Jun 2011; **83**:245 122, doi:10.1103/PhysRevB.83.245122. URL `http://link.aps.org/doi/10.1103/PhysRevB.83.245122`.
11. Rostgaard C, Jacobsen KW, Thygesen KS. Fully self-consistent gw calculations for molecules. *Phys. Rev. B* Feb 2010; **81**:085 103, doi:10.1103/PhysRevB.81.085103. URL `http://link.aps.org/doi/10.1103/PhysRevB.81.085103`.
12. Yan J, Jacobsen KW, Thygesen KS. Optical properties of bulk semiconductors and graphene/boron nitride: The bethe-salpeter equation with derivative discontinuity-corrected density functional energies. *Phys. Rev. B* Jul 2012; **86**:045 208, doi:10.1103/PhysRevB.86.045208. URL `http://link.aps.org/doi/10.1103/PhysRevB.86.045208`.
13. Larsen AH, Vanin M, Mortensen JJ, Thygesen KS, Jacobsen KW. Localized atomic basis set in the projector augmented wave method. *Phys. Rev. B* 2009; **80**:195 112.
14. Larsen AH. Efficient electronic structure methods applied to metal nanoparticles. PhD Thesis, Technical University of Denmark November 2011.
15. Enkovaara J, Romero NA, Shende S, Mortensen JJ. GPAW - massively parallel electronic structure calculations with Python-based software. *Procedia Computer Science (2011)* 2011; **4**:17–25.
16. Gropp W, Lusk E, Skjellum A. *Using MPI, 2nd Edition: Portable Programing with the Message Passing Interface*. The MIT Press, 1999.
17. Gropp W, Lusk E, Skjellum A. *Using MPI-2: Advanced Features of the Message-Passing Interface*. The MIT Press, 1999.
18. IBM Blue Gene team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development* 2008; **52**:199–220.
19. Baumeister P. *Real-Space Finite-Difference PAW Method for Large-Scale Applications on Massively Parallel Computers*, *Reihe Schlsseltechnologien / Key Technologies*, vol. 53. Foschungszentrum Jlich GmbH Zentralbibliothek, Verlag: Jlich, 2012. URL `http://juser.fz-juelich.de/record/128376`, schriftenreihen des Forschungszentrum Jlich; Zugl.: Aachen, RWTH, Diss., 2012.
20. Bottin F, Leroux S, Knyazev A, Zérah G. Large-scale *ab initio* calculations based on three levels of parallelization. *Computational Materials Science* 2008; **42**(2):329 – 336, doi:10.1016/j.commatsci.2007.07.019.
21. Hutter J, Curioni A. Dual-level parallelism for *ab initio* molecular dynamics: Reaching teraflop performance with the CPMD code. *Parallel Computing* 2005; **31**:1.
22. Hutter J, Curioni A. Car-parrinello Molecular Dynamics on Massively Parallel Computers. *ChemPhysChem* 2005; **6**:1788.
23. Gygi F. Architecture of Qbox: A scalable first-principles molecular dynamics code. *IBM journal of Research and Development* January/March 2008; **52**(1/2):1.
24. Giannozzi P, Cavazzoni C. Large-scale computing with Quantum ESPRESSO. *Nuovo Cimento C* 2009; **32**:49.
25. Bylaska EJ, Glass K, Baxter D, Baden SB, Weare JH. Hard scaling challengers for *ab initio* molecular dynamics capabilities in NWChem: Using 100, 000 cpus per second. *Journal of Physics: Conference Series* 2009; **180**:012 028.
26. Wang LW. Https://hpcrd.lbl.gov/ linwang/PEtot/PEtot.html.
27. Kent PRC. Computational challenges of large-scale long-time first-principles molecular dynamics. *Journal of Physics: Conference Series* 2008; **125**:012 058, doi:10.1088/1742-6596/125/1/012058.
28. Kresse G, Furthmüller J. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B* 1996; **54**:11 169–11 186.
29. Wood DM, Zunger A. A new method for diagonalising large matrices. *J. Phys. A: Math. Gen.* 1985; **18**(9):1343.
30. Payne MC, Teter MP, Allan DC, Arias TA, Joannopoulos JD. Iterative minimization techniques for *ab initio* total-energy calculations: molecular dynamics and conjugate gradients. *Rev. Mod. Phys.* Oct 1992; **64**(4):1045–1096.
31. Feng YT, Owen DRJ. Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems. *Int. J. Num. Meth. in Engineer.* 1996; **39**(13):2209–2229.
32. Davidson ER. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comp. Phys.* 1975; **17**:87–94.
33. Stewart G. Communication and matrix computations on large message passing systems. *Parallel Computing* 1990; **16**:27–40.
34. Dongarra J, van de Geijn R, Walker D. Scalability issues affecting the design of a dense linear algebra library. *Journal Parallel and Distributed Computing* 1994; **22**(3):523–537.
35. Hendrickson BA, Womble DE. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM Journal of Scientific and Statistical Computing* 1994; **15**(5):1201–1226.
36. Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, *et al.*. *LAPACK Users' Guide*. Third edn., Society for Industrial and Applied Mathematics: Philadelphia, PA, 1999.
37. Blackford LS, Choi J, Cleary A, D'Azevedo E, Demmel J, Dhillon I, Dongarra J, Hammarling S, Henry G, Petitet A, *et al.*. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics: Philadelphia, PA, 1997.
38. Thorton WS, Vence N, Harrison R. Introducing the MADNESS numerical framework for petascale computing. *Proceedings of the Cray Users Group* 2009; .

39. Nieplocha J, Palmer B, Tipparaju V, Krishnan M, Trease H, Apra E. Advances, applications and performance of the global arrays shared memory programming toolkit. *Int. J. High Perform. Comput. Appl.* 2006; **20**(2):203–231.
40. Blackford LS, Demmel J, Dongarra J, Duff I, Hammarling S, Henry G, Heroux M, Kaufman L, Petitet A, Pozo R, *et al.*. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Soft.* 2002; **28-2**:135–151.
41. *Engineering and Scientific Subroutine Library (ESSL)*. URL `http://www-03.ibm.com/systems/software/essl/`.
42. Solomonik E, Demmel J. Communication-optimal parallel 2.5d matrix multiplication and LU factorization algorithms. *Technical Report UCB/EECS-2011-10*, University of California, Berkeley Feb 2011.
43. M D Schatz JP, Van De Geijn RA. Scalable universal matrix multiplication algorithms: 2d and 3d variations on a theme. *Technical Report*, University of Texas at Austin 2012. URL `http://www.cs.utexas.edu/users/flame/pubs/SUMMA2d3dTOMS.pdf`.
44. Malony A, Shende S. Performance Technology for Complex Parallel and Distributed Systems. *Distributed and parallel systems: from instruction parallelism to cluster computing* 2000; :37–46.
45. Shende S, Malony A, Cuny J, Lindlan K, Beckman P, Karmesin S. Portable Profiling and Tracing for Parallel Scientific Applications using C++. *Proceedings 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98)*, 1998; 134–145.
46. Shende S, Malony A, Ansell-Bell R. Instrumentation and Measurement Strategies for Flexible and Portable Empirical Performance Evaluation. *Proceedings Tools and Techniques for Performance Evaluation Workshop, PDPTA*, vol. 3, CSREA, 2001; 1150–1156.
47. Shende S, Malony AD. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* Summer 2006; **20**(2):287–311. URL `http://www.cs.uoregon.edu/research/tau`.
48. Sosa C, Knudson B. *IBM System Blue Gene Solution: Blue Gene/P Application Development*. IBM Redbooks, 2009.
49. Foster I. *Designing and Building Parallel Programs*. Addison Wesley, 1995.
50. Chambers JM, Cleveland WS, Tukey PA, Kleiner B. *Graphical Methods for Data Analysis*. Wadsworth, 1995.
51. Kumar S, Dozsa G, Almasi G, Heidelberger P, Chen D, Giampapa ME, Blocksome M, Faraj A, Parker J, Ratterman J, *et al.*. The Deep Computing Messaging Framework: generalized scalable message passing on the Blue Gene/p supercomputer. *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, ACM: New York, NY, USA, 2008; 94–103, doi:10.1145/1375527.1375544.
52. Tisseur F, Donagara J. A parallel divide and conquer algorithm for symmetric eigenvalue problem on distributed architectures. *SIAM J. Sci. Comput.* 1999; **6:20**:223–2236.
53. Sunderland AG. Parallel diagonalization performance on high-performance computers. *Parallel Scientific Computing and Optimization*, vol. 27, Čieges R, Henty D, Kagström B, Žilinskas J (eds.). Springer, 2009; 57–66.
54. Petschow M, Peise E, Bientinesi P. High-performance solvers for dense hermitian eigenproblems. *SIAM J. Scientific Computing* 2013; **35**.
55. Gygi F, Draeger EW, Schulz M, de Supinski BR, Gunnels JA, Austel V, Sexton JC, Franchetti F, Kral S, Ueberhuber CW, *et al.*. Large-scale electronic structure calculations of high-z metals on the bluegene/l platform. *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, ACM: New York, NY, USA, 2006, doi:10.1145/1188455.1188502. URL `http://doi.acm.org/10.1145/1188455.1188502`.
56. Kleis J, Greeley J, Romero N, Morozov V, Falsig H, Larsen A, Lu J, Mortensen J, Dułak M, Thygesen K, *et al.*. Finite size effects in chemical bonding: From small clusters to solids. *Catalysis Letters* 2011; **141**(8):1067–1071.
57. Li L, Larsen AH, Romero NA, Morozov VA, Glinsvad C, Abild-Pedersen F, Greeley J, Jacobsen KW, Nørskov JK. Investigation of catalytic finite-size-effects of platinum metal clusters. *Journal of Physical Chemistry Letters* 2013; **4**(1):222–226.