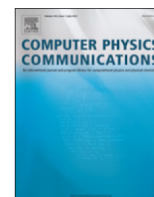
Computer Physics Communications 211 (2017) 8–15

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



GPU implementation of the linear scaling three dimensional fragment method for large scale electronic structure calculations

Weile Jia [a,b], Jue Wang [a], Xuebin Chi [a], Lin-Wang Wang [c,*]

[a] Supercomputing Center, Computer Network Information Center, Chinese Academy of Sciences, Beijing, China

[b] University of Chinese Academy of Sciences, Beijing, China

[c] Material Science Division, Lawrence Berkeley National Laboratory, Berkeley, USA

ARTICLE INFO

Article history:
Received 19 February 2016
Received in revised form
14 May 2016
Accepted 2 July 2016
Available online 9 July 2016

Keywords:
Electronic structure calculations
LS3DF
GPU

ABSTRACT

LS3DF, namely linear scaling three-dimensional fragment method, is an efficient linear scaling *ab initio* total energy electronic structure calculation code based on a divide-and-conquer strategy. In this paper, we present our GPU implementation of the LS3DF code. Our test results show that the GPU code can calculate systems with about ten thousand atoms fully self-consistently in the order of 10 min using thousands of computing nodes. This makes the electronic structure calculations of 10,000-atom nanosystems routine work. This speed is 4.5–6 times faster than the CPU calculations using the same number of nodes on the Titan machine in the Oak Ridge leadership computing facility (OLCF). Such speedup is achieved by (a) carefully re-designing of the computationally heavy kernels; (b) redesign of the communication pattern for heterogeneous supercomputers.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

To calculate the physical properties of many real materials, large systems containing thousands or tens of thousands of atoms are often necessary. For example, a 5 nm quantum dot can contain 5 thousand atoms. To describe the fluctuation of the dipole moment in $(CH_3NH_3)PbI_3$, a 20,000 atom system have been used [1]. An even larger system might be necessary to study the mechanical properties and the electronic structure consequence of dislocations. Although density functional theory (DFT) calculations are necessary for many of such problems, conventional DFT calculations cannot be used due to their $O(N^3)$ scaling of the system size N [2]. To solve this problem, various types of linear scaling methods have been developed. One particular approach of the linear scaling method is based on the divide-and-conquer strategy. In this approach, a global system is divided into many small (fragment) systems, and each fragment is solved quantum mechanically. After all the small fragment systems have been solved, their results are combined together to yield the result of the global system. Iteration loops can be used to ensure self-consistency between the global charge density and the fragment potentials. A major advantage of this divide-and-conquer approach is the possibility to use different computer process groups to solve different fragments. Since no communication is needed between different process groups for the computationally most expensive step (the quantum mechanical fragment calculation), the weak scaling of this algorithm can be extremely good. Thus, a dual linear scaling, one to the system size, one to the number of computer processes can be achieved.

Linear scaling three dimensional fragment (LS3DF) code is a Gordon Bell winning code based on a divide-and-conquer method [3]. It can be scaled efficiently to hundreds of thousands of processes. It has been used to calculate many nanostructure problems, including the dipole moment of nanorod [4]; the localized state in random alloy; the effects of $MoSe_2/MoS_2$ Moire's pattern [5]; the ferroelectric vortex structure [6]; the electronic structure of disordered $(CH_3NH_3)PbI_3$ [1]. Nevertheless, based on CPU, even though tens of thousands of processes (e.g., 60,000 processes) have been used, such calculations can often take several hours. This makes such calculations computationally expensive.

Graphic process unit (GPU) has been a main approach to accelerate the large-scale computation, especially to reach exascale computing. With 2048 arithmetic processing cores for each GPU card, and with a single instruction multiple data (SIMD) paradigm, GPU has proved to be naturally suitable for many scientific computation. Recently, we have used GPU to significantly speedup the standard plane wave pseudopotential (PW-PP) DFT

* Corresponding author.
E-mail addresses: jiawl@sccas.cn (W. Jia), wangjue@sccas.cn (J. Wang), chi@sccas.cn (X. Chi), lwwang@lbl.gov (L.-W. Wang).

http://dx.doi.org/10.1016/j.cpc.2016.07.003
0010-4655/© 2016 Elsevier B.V. All rights reserved.

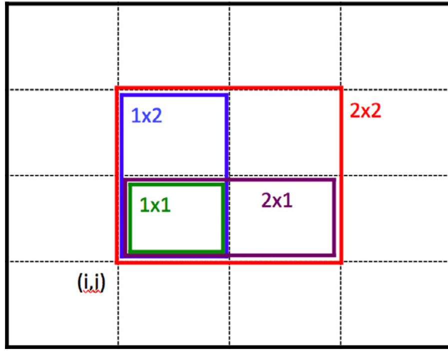


Fig. 1. The fragments used in the LS3DF calculation. The fragments have the sizes of 1×1, 2×1, 1×2 and 2×2 for a 2D illustration. For 3D, they should be $1 \times 1 \times 1$, $1 \times 1 \times 2$, $1 \times 2 \times 2$, $2 \times 2 \times 2$, etc. For each fragment (color), a buffer region also needs to be added to cancel the boundary effect. There are such fragments at each corner of the dashed line grid. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

calculations [7–9]. It is thus reasonable to ask whether that technique can also be used for LS3DF calculations. This is a particularly natural question since in LS3DF, the quantum mechanical wave functions of each fragment are solved using a PW-PP method. This lets the LS3DF having almost the same accuracy as the other conventional $O(N^3)$ scaling PW-PP codes, hence distinguish itself from other linearly scaling codes which often use localized orbitals. The same PW-PP method for the fragment provides the opportunity to take the advantage of the existing GPU PW-PP code to the LS3DF method.

There are however, other challenges unique to the LS3DF code. We found that, after the quantum mechanical calculations of the fragments are speeded up by GPU, the communication between the processor groups, which is used to patch together the charge density to obtain the global charge density, becomes the bottleneck. However, due to the use of GPU, the number of processes has been reduced by an order of magnitude (while the same number of nodes is used). This makes it possible to redesign the communication scheme, which has led to our final speedup.

The rest of the paper is organized as follows. Section 2 introduces the LS3DF algorithm, mainly on the conjugate gradient method and MPI communication. Section 3 describes the step-by-step speedup of the Kohn–Sham equation solution. Section 4 describes the GPU LS3DF MPI communication pattern. Section 5 shows the testing results and our discussion. Finally, Section 6 presents our conclusions and future work.

2. The LS3DF algorithm

LS3DF is based on the nearsightedness of the quantum mechanical effects [10], which means the effect of quantum mechanical is short range. As a result, one can divide a system into small parts, and the quantum mechanical effects at the center of one part will not be influenced by the atomic situations of the other parts. Nevertheless, the classical electrostatic potential is long ranged, thus has to be solved by the global Poisson equation. In LS3DF, a system is divided into many small fragments, and each fragment is solved for its electron wave functions, the quantum mechanical energy of the global system can be obtained through the summation of these fragment results. The LS3DF deploys a special division and patching scheme to cancel out the effects of the artificial boundary, as illustrated in Fig. 1. Both positive and negative fragments are used. As a result, the boundary effects from different fragments will be canceled out, and what left is one copy of the original system at the center of the fragments. The details of the LS3DF formalism can be found in Ref. [11].

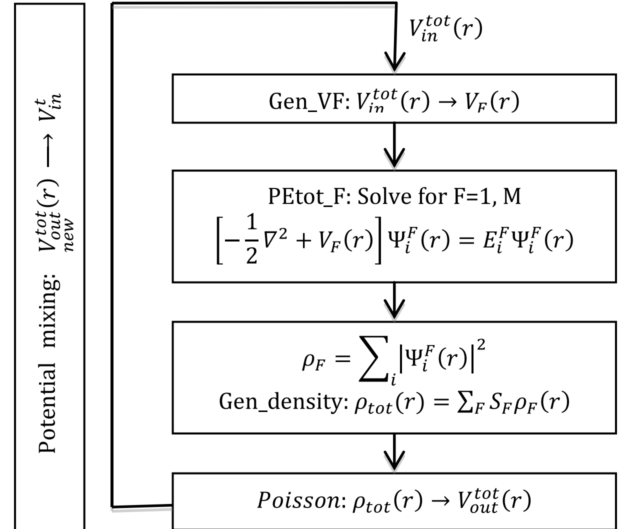


Fig. 2. The LS3DF self-consistent field flow chart. M is the number of fragments within one MPI process.

A flow chart of a LS3DF self-consistent field calculation is depicted in Fig. 2. For a given global input potential $V(r)$, the potentials of each fragment system $V_F(r)$ is generated. This $V_F(r)$ equals to $V(r)$ for the r within this fragment domain F, plus an additional passivation potential $\Delta V_F(r)$. After $V_F(r)$ is obtained, the fragment wave functions $\{\psi_i^F\}$ are solved based on the fragment Kohn–Sham equation: $H^F\psi_i^F = \varepsilon_i^F\psi_i^F$. This will be done by the conjugated gradient method to be described below. After $\{\psi_i^F\}$ are obtained, the fragment charge density $\rho_F(r)$ will be calculated as $\rho_F(r) = \sum_i \left|\psi_i^F(r)\right|^2$. The global charge density $\rho_{tot}(r)$ is calculated from all the $\rho_F(r)$ as $\rho_{tot}(r) = \sum_F S_F\rho_F(r)$, here $S_F = +/-$ is the sign of the fragment. After $\rho_{tot}(r)$ is obtained, the global Poisson equation is solved using in reciprocal space using a global FFT. The global Kohn–Sham potential $V(r)$ is then calculated using density functional theory. Next, a potential mixing scheme will be used to mix the output potentials from previous self-consistent field (SCF) iteration steps, the new mixed potential will be used as the input potential for next iteration run. This will ensure the SCF convergence. Usually, a Pulay mixing together with Kerker mixing are used. For a conventional system (e.g., semiconductors), even with thousands of atoms, typically 30–40 iteration steps are enough to fully converge the problem.

In the calculations of LS3DF, the MPI processes are divided into groups, as illustrated in Fig. 3. Each process group will calculate a few fragments quantum mechanically. Note that the fragments are distributed among the process groups to reach an optimal workload. For the fragment calculation, the wave functions are G-distributed (G-space parallel, as will describe in Section 3.1) within each process group [12]. The only global calculation is to solve the Poisson equation after the global charge density is obtained through the patching of the fragment charges. However, the global Poisson equation and the related FFT are only carried out within a subset of the whole process.

2.1. All-band CG

In the above LS3DF procedure, one of the most time consuming step is the solution of the fragment wave functions based on the fragment Kohn–Sham equation $H^F\psi_i^F = \varepsilon_i^F\psi_i^F$. There are different algorithms to solve this Kohn–Sham equation. Here, we will concentrate on the all-band conjugate gradient method (AB-CG), which is implemented in the LS3DF code. The AB-CG flow chart is shown in Fig. 4. It solves the Kohn–Sham equation $H^F\psi_i^F = \varepsilon_i^F\psi_i^F$

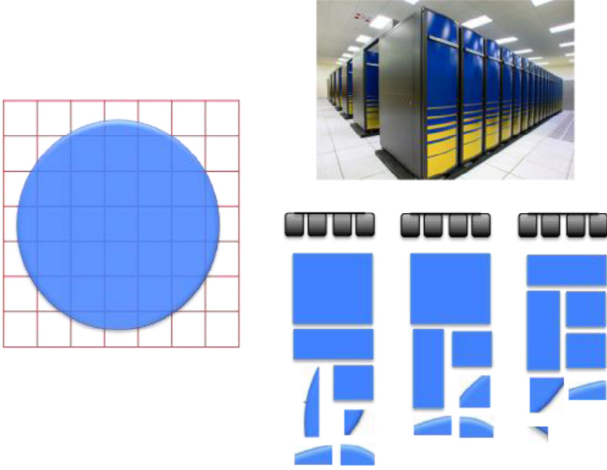


Fig. 3. An illustration of the fragments distribution onto supercomputer. Left part illustrates the calculated system (blue) and fragments divided by the red lines. Right part shows the distribution of the fragments (blue) over computing nodes (black). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$P_i^F = H^F\psi_i^F - \varepsilon_i^F\psi_i^F \quad \text{Hpsi*}$$
$$h^F(i,j) = \langle\psi_i^F|H|\psi_j^F\rangle \quad \text{Sub_diag*}$$
$$\rightarrow P_i^F = A\left(P_i^F - \frac{\lambda_i}{\lambda_i^0}P_i^{F0}\right) \quad \text{Precond. CG step}$$
$$P_i^F = P_i^F - \textstyle\sum_{j=1,i}\langle P_i^F|\psi_j^F\rangle \quad \text{Projection*}$$
$$P_i^F = H^F\psi_i^F - \varepsilon_i^F\psi_i^F \quad \text{Hpsi*}$$
$$\psi_i^F = \psi_i^F\cos\theta_i + P_i^F\sin\theta_i \quad \text{Line minimize}$$
$$\psi_i^F = \psi_i^F - \textstyle\sum_{j<i}\langle\psi_i^F|\psi_j^F\rangle \quad \text{Orth*}$$
$$h^F(i,j) = \langle\psi_i^F|H|\psi_j^F\rangle \quad \text{Sub_diag*}$$

Fig. 4. The all-band conjugate gradient (AB-CG) method for $H^F\psi_i^F = \varepsilon_i^F\psi_i^F$. The asterisk sign indicates the time consuming steps.

iteratively. First, a $P_i^F = H^F\psi_i^F - \varepsilon_i^F\psi_i^F$ is calculated from the input wave functions. Then, a subspace diagonalization is carried out based on the complex matrix $\langle\psi_i^F|H^F|\psi_j^F\rangle$. After this, conjugate gradient steps with line minimization are calculated. During this iteration, the computationally most expensive steps include the following: $P_i^F = H^F\psi_i^F$ (Hpsi); projection of residual vector P_i^F from the $\{\psi_i^F\}$ subspace $P_i^F = P_i^F - \varepsilon_i^F\psi_i^F$ (Projection); and orthogonalization (Orth) among $\{\psi_i^F\}$. The text in the bracket is used to denote the steps shown in Fig. 4. At the end of AB-CG, another subspace diagonalization is carried out. Overall, this AB-CG algorithm is almost the same as the one in the stand-alone PW-PP code PEtot [13]. The PEtot AB-CG subroutine has already been implemented in GPU. Here, we port that GPU AB-CG code into the LS3DF code. Besides the change of variables, global variable inputs and MPI communication domains, there are issues about load balance, as well as the efficiency of the GPU calculation for the different sized fragments. Since the sizes of the fragments are limited in a LS3DF calculation (e.g., the maximum is about 200 atoms), this provides the opportunity to optimize the GPU calculation, e.g., by moving all the wave functions into the GPU memory.

2.2. Data communication between different process groups

After the AB-CG is implemented with GPU, the next bottleneck is the data communication between the process groups. This is mainly for gathering the $\rho_F(r)$ together to generate the global $\rho_{tot}(r)$ and to distribute the potential $V(r)$ to generate $V_F(r)$. The later part is essentially the reverse process of the former part. Note that, the global $\rho_{tot}(r)$ and $V(r)$ are distributed within a subset P_G of the whole processes, not the whole process group. This is because the FFT is only efficient over a finite number of processes. If the FFT grid of the global box is $n1 \times n2 \times n3$, we usually take $n1$ as the number of process in P_G. This ensures that each process in P_G has one slide ($n2 \times n3$) of the real space data set. This will be important in designing our data communication pattern.

In the original CPU implementation, due to the large number of CPU process used, the communications are carried out in two steps. At the first step, the distributed $\rho_F(r)$ within a process group will be collected by one "head" process in that group. Then, this head process in this group will communicate to the P_G process in the P_G group. However, due to the memory constraint, the head process cannot pre-collect all the real space data in its group, and then communicate to P_G whenever it is needed. Instead, the data needed by one P_G are collected by the head process on the flight. This has significantly increased the number of communication, and makes the communication fragmented and expensive.

For a GPU run on the Titan machine, only one CPU is used on one node (the other 15 CPUs will be kept idle), which is tight to one GPU on that node. As a result, the number of process has been significantly reduced. Because of this, we have redesigned the communication pattern. Instead of two step communication process, we have used one step communication, let all the processes communicate directly to P_G. The result is order of magnitude improvement for the communication time.

3. GPU implementation of the AB-CG code

3.1. CPU parallelization of the AB-CG

Before discussing the GPU implementation, we like first to discuss the original CPU parallelization of the AB-CG subroutine in LS3DF. For LS3DF calculations, the fragments are treated as isolated systems, thus only the Gamma point is used, thus there is no k-point parallelization [14]. The first possible parallelization in AB-CG is the band index parallelization. It is the parallelization over the band index 'i' in wave functions $\{\psi_i^F\}$. In a stand-alone DFT code, each group of process stores a block of wave functions. If the whole wave function ψ_i for a given i is stored within one process, then MPI communication will not be needed in the calculation of Hpsi. However, the calculation of the wave function overlaps, like $\langle\psi_i|H|\psi_j\rangle$, $\langle P_i|\psi_j\rangle$, $\langle\psi_i|\psi_j\rangle$, will require the communication of $\{\psi_i^F\}$. If these wave functions are passed around in a round robin style, this communication will be time consuming. Because of this, the band index parallelization is not used in the LS3DF calculation. The second choice for parallelization is the G-space parallelization. In this parallelization, the G-space coefficients of the wave function are distributed among processes within a fragment process group. In a plane wave calculation, the wave functions are expanded by the plane wave basis set. Each plane wave function corresponds to one reciprocal lattice points of the real space periodic box. These lattice points are within a G-space sphere in the FFT grid. On the other hand, the real space FFT grid of a fragment is a full 3D box with $n_{1F} \times n_{2F} \times n_{3F}$ points. G-space parallelization scheme does the FFT within the n_nodes processors. The FFT is optimized for the spherical G-space data by load balancing and minimizing the communication between processors [15]. The nonlocal potential projection operator $\sum_l |\phi_l\rangle\langle\phi_l|$ in LS3DF is calculated in G-space within a sphere for each atom [16].

3.2. ZGEMM to CUBLAS_ZGEMM

Our GPU implementation starts with the computationally intensive kernels in the AB-CG algorithm: calculate the overlaps matrix in Sub_diag, Projection and Orth steps shown in Fig. 4. In the CPU implementation, these operations are carried out by calling the matrix–matrix multiplication BLAS-3 ZGEMM subroutine. In the GPU implementation, they can be easily replaced with a CUBLAS-3 subroutine CUBLAS_ZGEMM. Since the book-keeping in CPU is still needed, each time we calculate the overlap matrix, the corresponding wave functions $\{\psi_i^F\}$, $\{P_i^F\}$, and $\{W_i^F \equiv H^F\psi_i^F\}$ will be copied from the CPU to the GPU. The resulting overlap matrix has to be copied back from the GPU to the CPU. After that, an MPI_Allreduce within the fragment group processes is used to sum over the $m \times m$ matrix, where m is the number of wave functions for this fragment. The obtained matrix h, S will be diagonalized or Cholesky decomposed in CPU, and the diagonalization (or decomposed) matrix will be sent back to the GPU, to be followed with wave function rotations (or projections for P_i). The wave function rotation is carried out using CUBLAS_DGEMM or CUBLAS_ZGEMM. In our test, by moving the matrix–matrix multiplication into GPU, we could obtain 4 times speedup comparing one GPU with single CPU core. The CPU–GPU wave function copies due to the book-keeping of the wave functions in CPU could be further reduced by moving all computation kernels into GPU.

3.3. GPU hybrid parallelization of the AB-CG

Similar to the stand-alone PEtot GPU implementation [7], a G-space and band-index hybrid parallelization scheme has been adopted in a GPU LS3DF code. A G-space parallelization is used in calculating overlap matrix like Sub_diag, Projection and Orth (section above), and a band-index parallelization is used to calculate Hpsi, as shown in Fig. 5. In this way, when Hpsi is calculated, the whole wave function is stored in a single CPU/GPU computing unit. Wave function book-keeping is still in CPU with G-space parallelization (each CPU hold all the wave function index i, but only part of the G-space coefficients). An MPI_Alltoall communication within the fragment process group is needed in this hybrid parallelization scheme to transpose the wave functions from G-space parallelization (for overlap matrix calculation) to band-index parallelization (for Hpsi calculation). After this data transpose, each CPU/GPU computing unit will hold a subset of the band index i, but with the full wave function for each i within the subset. After mapping the G-space coefficients within the sphere into a full FFT grid and padding the outside of the sphere with zeros, a standard 3D CUFFT (on a single GPU) will be applied to the wave functions. This is used for the Hpsi calculation. The FFTs are calculated in a band-by-band manner in order to save GPU memory. Our tests show that the MPI_Alltoall takes up to 50% of the total time to calculate Hpsi starting from the CPU book-keeping G-space parallelization for fragments with about one hundred atoms.

3.4. Moving most calculations into the GPU

The third step of GPU implementation of AB-CG is to move all the computations into GPU. This is needed because the CPU–GPU data movement is rather slow compared with its enormous computing power. For example, K20X GPU has a peak performance of 1.31 teraflops in terms of double precision computing power, while PCI-Express data movement is only 16 GB/s. This means that more computation should be moved into GPU to reduce the CPU–GPU memory copy operation. In this step, the wave functions are kept in the GPU, and book-keeping of the wave functions on CPU is no longer needed. Since K20X GPU has a memory of 6 GB, it is capable of holding more than 200 full wave functions (each wave function is no more than 30 MB) for the largest fragments. Wave functions updates in Sub_diag, Projection and Orth are directly performed inside GPU. The only part that still requires wave function CPU–GPU copy operation is before and after the MPI_Alltoall of the $H^F\psi_i^F$ operation. Diagonalization of matrix $h(i,j)$ using zheev and Cholesky decomposition on the overlap matrix $\langle\psi_i|\psi_j\rangle$ using zpotrf are also examined. While this part does not scale with the number of processors, we tested different libraries and chose ELPA_SOLVE_EVP for matrix diagonalization and GPU MAGMA_ZPOTRF and CUBLAS_ZTRSM for Cholesky decomposition and wave function rotations, respectively. Other parts of the code, e.g., precondition and line minimization are also moved into GPU by hand-coded CUDA code, which also gives us a gain in the speed. Overall, keeping wave functions inside GPU throughout AB-CG gives us improvements in both computation and CPU–GPU memory copy. Our tests show that this part could improve the speed by a factor of 2.

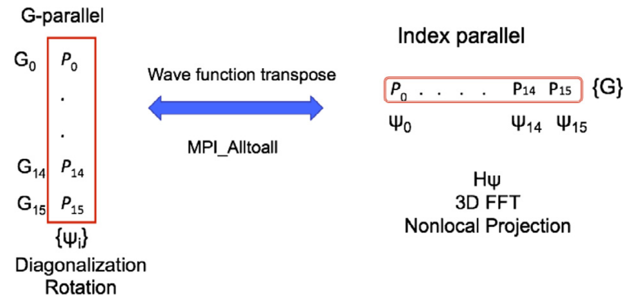


Fig. 5. G-space and band-index hybrid parallelization scheme in LS3DF GPU code. P_m denote the process, G_m the G-space portion, and ψ_i the full wave function of index i. The book-keep is at the left hand side, while when Hpsi is calculated, the data of the wave functions are transposed into the right hand side.

3.5. MPI communications and mix precision calculations

The MPI_Alltoall within the fragment process group is a bottleneck in the GPU AB-CG method, yet there is no easy way to avoid the wave function transpose in the hybrid parallelization scheme. To reduce the communication between MPI tasks, a compression algorithm is introduced in the GPU AB-CG. In the AB-CG algorithm, although not apparent in Fig. 4, the $H^F\psi_i^F$ is actually calculated by $H^FP_i^F$, where P_i^F is the wave function residual $P_i^F = H^F\psi_i^F - \varepsilon_i^F\psi_i^F$ from the last iteration and it is always very small. The purpose of the AB-CG algorithm is to minimize the residual P_i^F. Typically, P_i^F is reduced by approximately a factor of 10 in one CG step. Thus, it is not necessary to keep the residual P_i^F in double precision. Note that P_i^F is in double complex precision in the CPU implementation. In the GPU implementation, we use 4 bytes instead of 16 bytes for the residual P_i^F. Nine binary digits are used to represent the numerical amount, and 6 binary digits are used to represent its exponent. The algorithm is shown in Algorithm 1. This significantly reduced the MPI_Alltoall data size to 25% of its original amount. Note the residual is compressed in the GPU and it takes little time. Furthermore, the precision reduction makes it possible to use single precision in the GPU calculation. Double precision is used in all other parts except the Hpsi calculation of the residual P_i^F. FFT and nonlocal potential calculations are both evaluated in single precision operations in GPU. The compression does not affect the final result in precision, nor does it affect the convergence of the AB-CG algorithm as shown in Fig. 6.

Algorithm 1. The wave function residual data compression.

```
Input: wave function residual R^F
Output: wave function residual R^F
    R_i^F ≡ (−1)^{S_i} × f_i × 2^{e_i}
    m = max (e_i)
    g_i = ⌊f_i × 2^9⌋/2^9
    d_i = max (64, m − e_i)
    R_i^F = (−1)^{S_i} × g_i × 2^{m−d_i}
```

3.6. Wave function occupation

Another computationally intensive kernel is to occupy the wave function to get the charge density as expressed in $\rho_F = \sum_i \left|\Psi_i^F(r)\right|^2$. The issue here is that the original wave functions are stored in G-space with G-space parallelization data distribution. We need to do FFT over the G-space wave functions to get the real space wave functions. In the CPU implementation, parallel FFT is calculated among the G-parallel processors. However, in the GPU code, an MPI_Alltoall is needed to transpose the G-space parallel data distribution to band index parallelization, so that one CPU/GPU computing unit has a full wave function. After the FFT, in order to sum up the real space wave functions distributed in different CPU/GPU units, an MPI_ReduceScatter on the real space charge density $\rho_{tot}(r)$ needs to be performed. After the partial charge density ρ_F is obtained, an MPI communication is needed to patch the fragmented charge density into global charge. Then the Poisson equation is solved (via a global FFT) to get the potential for the next SCF iteration.

4. Global MPI communication in the LS3DF code

In this section, we discuss the improvement of the global MPI communication scheme for $\rho_F(r)$ and $V_F(r)$. Although these communications are done purely on CPU, the reduction of the number of CPU process due to the use of GPU (only one CPU is used on one node) makes it possible to improve the communication scheme.

4.1. The original MPI communication scheme

For the LS3DF code, the computationally intensive parts are solving the Kohn–Sham equation $H^F\psi_i^F = \varepsilon_i^F\psi_i^F$ and occupation of the wave function. On the other hand, distributing the global potential $V_{in}(r)$ among different MPI tasks and getting the total density $\rho_{tot}(r)$ from the fragmented charge density take most of the time in terms of global MPI communications. In a typical CPU LS3DF SCF step, the above communication parts take up to 20% of the total time.

In the SCF iteration shown in Fig. 2, distributing the global $V_{in}(r)$ and getting the total density $\rho_{tot}(r)$ are exactly opposite operations. Here, we will only examine the total density gathering operation. After the wave function calculation in AB-CG, the ρ_F is calculated within each MPI processor group using $\rho_F(r) = \sum_i \left|\Psi_i^F(r)\right|^2$. The global $\rho_{tot}(r)$ is then calculated through $\rho_{tot}(r) = \sum_F S_F\rho_F(r)$, $S_F = +/-$ is the sign of the fragment. Because the CPU LS3DF can scale to hundreds of thousands cores, a two-layer hierarchical communication scheme was designed specifically to avoid direct global communication. It begins with dividing the global charge density $\rho_{tot}(r)$ into slices. Then all MPI processor groups (as shown in Fig. 3) first collect the charge density of that particular slice within the group, and give them to a lead process within the group. Because one MPI processor group can hold several fragments, this part is actually a summation over different fragments. Next, an MPI_allreduce among different fragment groups is needed to get the slice of the global charge

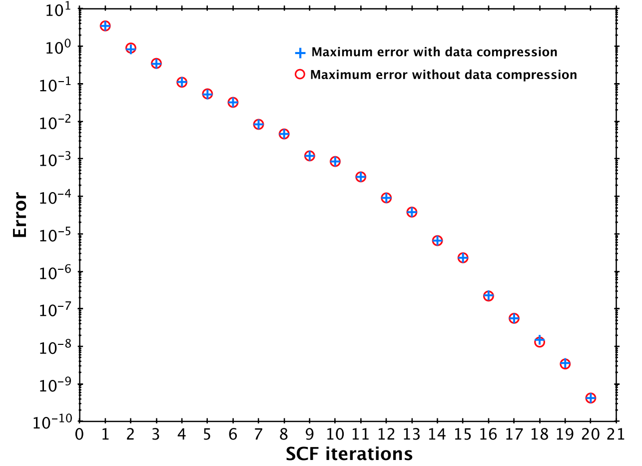


Fig. 6. Comparison of the maximum errors of the 512 atom GaAs system eigenstates for 20 AB-CG steps with and without data compression. This system has 1024 electrons and the FFT size is 128^3. Each SCF step is consisted of 4 line minimization. The vertical axis is in logarithmic scale with a base 10. Note that the data compression does not change the convergence of the SCF.

density $\rho_{tot}(r)$ and give it to one process within the P_G group (to calculate the global Poisson equation). After this communication, the Poisson equation will be solved via parallel FFT within the P_G group to get the $V_{tot}^{out}(r)$ for the next SCF iteration.

4.2. The modified MPI communication scheme

The CPU two-layer communication scheme limits the communication within fragment process groups, and global communication is avoided. The disadvantage however is the increased number of MPI communication calls since for each slide of the $\rho_{tot}(r)$, the charge densities need to be collected from different processes within a fragment group in an on-the-flight fashion. This two level communication pattern might be suitable when the total number of process is extremely large (e.g., 60,000 processes), where a direct all-to-all communication can overwhelm the network. In the GPU LS3DF code, however, the number of total processes (the total MPI tasks) has been significantly reduced (by 16 times). In the Titan machine, much like other typical heterogeneous supercomputer, one computing node is equipped with 16 CPU cores, but only one GPU card. For the CPU code, if we use all computing power of Titan, we will have 299,008 MPI tasks. On the other hand, in a GPU run we only need only 18,688 MPI tasks (one CPU core binding to one GPU card). Since GPU code uses much less MPI tasks, one can attempt to use a simple communication pattern.

A one-layer communication pattern is used in our GPU LS3DF code. The idea is that in order to get the total density, we use a point-to-point MPI communication to send the fragmented charge density $\rho_F(r)$ from a process within one fragment process group directly to the corresponding MPI task within the P_G group that holds the $\rho_{tot}(r)$. Since the total charge density is distributed among n_1 MPI processors (n_1 being the first dimension of the global FFT grid) within the P_G group and each of the n_1 processor will hold

Algorithm 2. MPI handshake to get the charge density global index.

```
MPI senders:
    For each fragment that reside in current MPI process:
      For each charge density data point:
        Calculate the global index (x,y,z) of the data point within the global grid
        Packing the global index into the buffer [x].
      Endfor
    Endfor
    For x < n1:
      Send buffer[x] to MPI rank x
    Endfor

//only the first n1 MPI processes receive.
MPI receiver:
    For j < nprocessors:
      Receive the buffer[j] from rank j.
      Unpack the buffer and construct the global index from all received buffer.
    Endfor
```

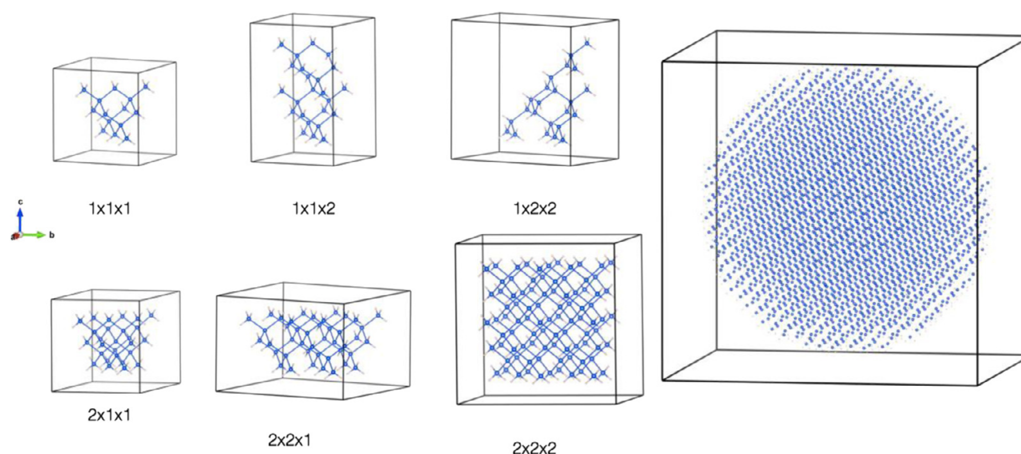


Fig. 7. The illustration of different size of fragments and the entire 3877 atom Si quantum dot system. Blue dots represent the Si atoms and orange ones represent the H atoms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

a slide of $n_2 \times n_3$ data, a mapping index array is first generated in order to map the fragment charge to the global charge. In the implementation, two handshakes are used to setup the mapping array. In the first handshake, a single number is send from each processor to the n_1 processors, indicating the total number of the communicational data in the following handshake. The mapping array is obtained by executing Algorithm 2. Note that the mapping array is calculated before the SCF iteration begins and it does not need to be recalculated during the SCF steps.

Once the global mapping array is obtained, the Gen_density procedure of Fig. 2 becomes straightforward. In MPI senders, fragmented charge density data is packed and sent in the same manner as in Algorithm 2. Next the $n1$ MPI receiver (within the P_G group) will receive the data, and then a mapping procedure is executed to map the fragmented charge density to the global charge density. Note that the handshake is one time only, and only the charge density point-to-point communication is necessary in the SCF calculation. Compared to the CPU two-layer communication pattern, the direct communication for heterogeneous architecture is more efficient. It suits the current heterogeneous supercomputers.

5. Results and discussions

We have used two nanosystems to carry out a comparison between the original CPU version of the LS3DF code and the new GPU version of the code. The first test runs on Titan supercomputer with 24,000 CPU processors for the CPU code, and 1500 MPI tasks for the GPU code. The second test runs with 55,296 CPU MPI processes, and 3456 MPI tasks for the GPU code. Note that for each test, CPU and GPU use the same number of computing nodes.

The first system is a Si quantum dot passivated by H atoms. The system has 3877 atoms, 13,024 electrons. The box size is [102.6, 102.6, 102.6] Å, the global FFT grid is [480, 480, 480]. The largest $2 \times 2 \times 2$ fragment contains 199 atoms with 472 electrons. With the vacuum buffer, the largest $2 \times 2 \times 2$ fragment has a grid point of [144, 144, 144]. The energy cut off for the plane wave basis set is 489.8 eV. Fig. 7 shows the Si quantum dot system as well as a few prototypical fragments. The second system is $CaTiO_3$ perovskite system. The system has 8640 atoms, 41,472 electrons. The box size is [86.32, 86.32, 86.32] Å, the global FFT grid is [384, 384, 384]. The largest $2 \times 2 \times 2$ fragment contains 95 atoms with 288 electrons. With the vacuum buffer, the largest $2 \times 2 \times 2$ fragment has a grid point of [128, 128, 128]. The energy cut off for the plane wave basis set is 816.3 eV.

The computation and the tests are carried out on the Titan supercomputer at the Oak Ridge Leadership Computing Facility. The Titan machine has 18,688 computing nodes; each node is equipped with one 16-core AMD Opteron 6274 CPU and one Kepler K20X GPU. The Kepler K20X GPU has a peak performance of 1312 Gflops, so totally the accelerators contribute 24.5 petaflops of the Titan 27 petaFlops peak performance. The computing nodes are connected through Cray's high-speed Gemini network.

Table 1
The partial and total computational time (s) and the GPU speedup compared to CPU for two testing systems. Note that the tests are performed on Titan supercomputer with the same number of computing nodes. Each node has 16 AMD operon cores and one GPU card. So the comparison is one K20X GPU card against 16 CPU cores.

		Process	Gen_VF	AB-CG occupy	Gen_density	Poisson	Total
3877	CPU	24,000	200	1368	307	33	1908
Atom	GPU	1,500	23	236	15	28	302
Si	Speedup		8.7x	5.8x	20x	1.2x	6.3x
8640	CPU	55,296	220	7433	340	15	8008
Atom	GPU	3,456	82	1559	91	10	1742
$CaTiO_3$	Speedup		2.7x	4.7x	3.7x	1.5x	4.5x

It takes 18 SCF steps to reach convergence for the 3877 atom Si quantum dot system using both CPU and GPU on 1500 computing nodes. The CPU code takes 1908 s for 18 SCF steps, each step taking 106 s. The GPU code takes 302 s for 18 SCF steps, each SCF step taking 16.8 s. Note that the GPU LS3DF code has the same convergence rate compared with CPU code, as shown in Fig. 8. The 8640 atom $CaTiO_3$ system is calculated on 2345 computing nodes on Titan supercomputer. It takes the CPU code 8008 s to finish 30 SCF steps, while the GPU code finishes the same 30 SCF iteration with 1742 s. The overall speedup is 4.5x (see Table 1).

The total computational time shows that the GPU code has a speedup of 4.5x to 6x. This can be divided into three parts: the AB-CG and Occupy part, the Gen_VF and Gen_density part, and the Poisson equation part. AB-CG and Occupy is the most computationally intensive part. As discussed in session 3, it takes 4 steps to move the AB-CG algorithm entirely into the GPU. However, the speedup is not ideal. First, in the SCF iteration, there are still parts residing in the CPU, e.g., to get the nonlocal G-space projector. The other reason is that for small fragments like $1 \times 1 \times 1$, which could contain only 5 atoms, there is not enough data operation to fully utilize the GPU computational power. In a real application scenario, such small $1 \times 1 \times 1$ fragments take a large proportion; further optimization is perhaps possible in the future work.

The Gen_VF and Gen_density parts are the global communication parts. As discussed before we use a single-layer MPI communication pattern instead of the two-layer communication pattern. The testing results show we have 8.6x and 20x speedup for Gen_VF and Gen_density, respectively. Note that these two procedures are the opposite MPI operations. The Gen_VF distributes total potential into fragments and Gen_density collects the fragmented charge density to get a total charge density. The total charge density is solved via Poisson solver to get the total potential $V(r)$. The main difference between Gen_VF and Gen_density are that Gen_VF sends data from a group of processors (P_G) to all the MPI processes, while Gen_density sends data from all MPI processes to the group of processors P_G. These two communications are un-symmetric, which leads to their slight time differences.

Another part is the Poisson solver. In the Si system, it takes 1.7 percent of the total time; while in the CaTiO3 system, it takes 0.2 percent of the total time. In the Poisson solver, the global FFT is solved in parallel within the n_1 processors in the P_G group ($n1$ is the first dimension of the global FFT). The GPU LS3DF uses the same Poisson solver as the CPU LS3DF code. The speedup comes from the fact that we use one MPI per node in the GPU code, thus the network contentions are avoided.

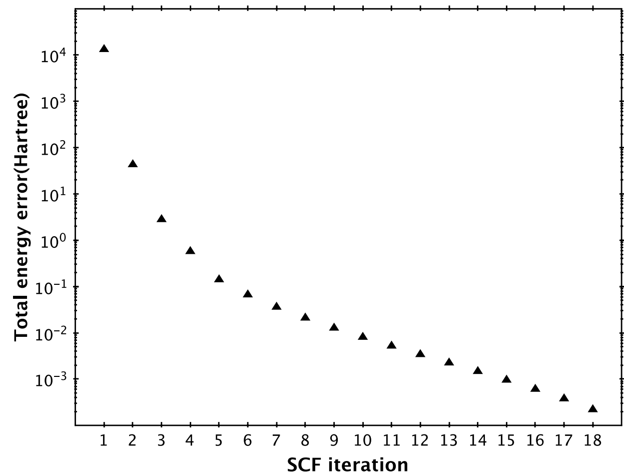


Fig. 8. The SCF convergence of the CPU and GPU LS3DF code for 3877 atom Si quantum dot system. Note that the GPU and CPU code convergence is the same. The vertical axis is in logarithmic scale with a base 10.

6. Conclusions and future work

In this paper, we presented our LS3DF GPU work on heterogeneous supercomputer. This code can calculate a system with thousands of atoms for SCF convergence within 5–25 min when enough GPU nodes are used. It is about 4.5–6 times faster than the corresponding CPU code. We have presented the detailed steps to speedup the code. This includes (1) a hybrid parallelization between G-space and band-index parallelization to speedup the FFT; (2) moving all the computationally heavy parts into GPU to reduce CPU–GPU memory copy operations; (3) a data compression algorithm to reduce the MPI_Alltoall communication; (4) using direct point-to-point MPI for global communication when patching up the charge density. Nanosystem electronic structure calculation can now be reduced from hours to minutes. For example, one SCF step of the 8640 atoms $CaTiO_3$ system (with 41,472 electrons) takes only about one minute.

Current GPU AB-CG and Occupy takes about 80% of the total computational time. One of our future works is to further speedup this kernel. The bottleneck is with the small fragments, as mentioned in session 5. Two ways could be used to further speedup this part, the first is moving other CPU parts, e.g., calculating the nonlocal projector, into GPU; the second is to use CUDA streams to further exploit the parallelization of the small fragments. In this paper, we have used one MPI per GPU. However, on the Titan supercomputer, one node is equipped with 16 CPU cores and 1 GPU card. In order to fully utilize the CPU part, one good programming model would be MPI/OpenMP/CUDA. Nevertheless, such a programming model would be a big challenge in the implementation, as we have moved the most computationally intensive tasks already into the GPU.

Acknowledgments

W.J. is supported by the China Scholarship Council under No. 201404910432. L.W.W is supported by the U.S. Department of Energy, SC/BES/MSED under the Contract No. DE-AC02-05CH11231 through the Material Theory project. Jue Wang is supported by the Hi-Tech Research and Development Program (863) of China No. 2015AA01A303, Natural Science Foundation of China under Grant No. 61303050, and the Youth Innovation Promotion Association, CAS(2015375). We used the computational resources of the Oak Ridge Leadership Computing Facility at the

Oak Ridge National Laboratory, which is supported by the Office of Science of the DOE under Contract No. DE-AC05-00OR22725, with computational time allocated by the Innovative and Novel Computational Impact on Theory and Experiment project.

References

[1] J. M, L.W. Wang, Nano Lett. 15 (2015) 248.
[2] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, J.D. Joannopoulos, Rev. Modern Phys. 64 (1992) 1045.
[3] L.W. Wang, B. Lee, H. Shan, Z. Zhao, J. Meza, E. Strohmaier, D. Bailey, Proc. 2008 ACM/IEEE Conf. Supercomp., ACM Gordon Bell, 2008, Artcle 65.
[4] S. Dag, S.Z. Wang, L.W. Wang, Nano Lett. 11 (2011) 2348.
[5] J. Kang, J. Li, S.S. Li, et al., Nano Lett. 13 (2013) 5485.
[6] Z. Gui, L.W. Wang, L. Bellaiche, Nano Lett. 15 (2015) 3224.
[7] Long Wang, Weile Jia, Xuebin Chi, Yue Wu, Weiguo Gao, Lin-Wang Wang, The International Conference for High Performance Computing, Networking, Storage, and Analysis, 2011, Published.
[8] W. Jia, Z. Cao, L. Wang, X. Chi, W. Gao, L.W. Wang, Comput. Phys. Comm. 184 (2013) 9.
[9] W. Jia, J. Fu, Z. Cao, L. Wang, X. Chi, W. Gao, L.W. Wang, J. Comput. Phys. 251 (2013) 102–115.
[10] W. Kohn, Phys. Rev. Lett. 76 (1996) 3168.
[11] L.W. Wang, Z. Zhao, J. Meza, Phys. Rev. B 77 (2008) 165113 [LBNL-63252].
[12] A. Canning, J. Shalf, L.W. Wang, H. Wasserman, M. Gajbe, Proceed. Parco09, Lyon France, 2009.
[13] L.W. Wang, PEtot code: https://hpcrd.lbl.gov/~linwang/PEtot/PEtot.html.
[14] A. Canning, L.W. Wang, A. Williamson, A. Zunger, J. Comput. Phys. 160 (2000) 29.
[15] A. Canning, J. Shalf, L.W. Wang, H. Wasserman, M. Gajbe, Proceed, Parco09, Lyon France, 2009.
[16] Z. Zhao, J. Meza, L.W. Wang, J. Phys.: Conds. Matt. 20 (2008) 294203.