Large Scale Plane Wave Pseudopotential Density Functional Theory Calculations on GPU Clusters

Long Wang
Supercomputing Center of Computer Network Information Center, Chinese Academy of Sciences
No. 4 South 4th Street, ZhongGuanCun, Beijing, China
008618611845735
wangl@sccas.cn

Yue Wu
School of Mathematical Sciences, Fudan University
220 Handan Road, Shanghai 200433, China
008613564955638
09210180026@fudan.edu.cn

Weile Jia
Supercomputing Center of Computer Network Information Center, Chinese Academy of Sciences
No. 4 South 4th Street, ZhongGuanCun, Beijing, China
008615510083754
jiawl@sccas.cn

Xuebin Chi
Supercomputing Center of Computer Network Information Center, Chinese Academy of Sciences
No. 4 South 4th Street, ZhongGuanCun, Beijing, China
008613910427213
chi@sccas.cn

Weiguo Gao
MOE Key Laboratory of Computational Physical Sciences, School of Mathematical Sciences, Fudan University
220 Handan Road, Shanghai 200433, China
008613761931985
wggao@fudan.edu.cn

Lin-Wang Wang†
Material Science Division, Lawrence Berkeley National Laboratory
One Cyclotron Road
Mail stop 50F
Berkeley, CA 94720
0015104865571
lwwang@lbl.gov

ABSTRACT
In this work, we present our implementation of the density functional theory (DFT) plane wave pseudopotential (PWP) calculations on GPU clusters. This GPU version is developed based on a CPU DFT-PWP code: PEtot, which can calculate up to a thousand atoms on thousands of processors. Our test indicates that the GPU version can have a ~10 times speed-up over the CPU version. A detail analysis of the speed-up and the scaling on the number of CPU/GPU computing units† (up to 256) are presented. The success of our speed-up relies on the adoption a hybrid reciprocal-space and band-index parallelization scheme. As far as we know, this is the first GPU DFT-PWP code scalable to large number of CPU/GPU computing units. We also outlined the future work, and what is needed to further increase the computational speed by another factor of 10.

Categories and Subject Descriptors

General Terms

Keywords
Electronic structure, First-principles, Density functional theory, Plane wave pseudopotential, GPU.

1. INTRODUCTION
Density functional theory (DFT) has become the workhorse for material science simulations [1]. It combines the accuracy for total energies and atomic structures, with relative ease for computation compared to high order quantum chemistry calculations. Nowadays, material science simulations have been divided into two categories. One is classical simulations, which use direct and approximated expressions based on the atomic coordinates for the total energy and atomic force of the system [2]. The other one is \textit{ab initio} simulation, which solves the quantum mechanical Schrodinger’s equation to get the total energy of a system, and no empirical or experimental parameter is ever used [3]. If affordable, the \textit{ab initio} simulation is the preferred way to carry out the simulation due to its high accuracy and reliability. Nowadays, when people talk about \textit{ab initio} method, they often mean the DFT method. However DFT calculations can be thousands of times more expensive than the classical simulations. In many large computer centers, like the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Lab, the DFT simulation consumes the most computer time among different computational methods and tasks [4]. Thus, how to speed up such DFT calculations is of paramount importance.

There are many numerical ways to carry out DFT calculations. It could be the full potential linearized augmented plane wave method (FLAPW) [5], the real space finite difference method [6], the real space finite elements method [7], the atomic orbital method [8], the wavelets method [9], and the plane wave pseudopotential (PWP) method [10]. The PWP method is the most mature and widely used method for DFT calculations. Arguably, there are more PWP DFT codes than any other methods [11], and the recent surge of \textit{ab initio} material science simulation is partially due to the maturity of such PWP DFT codes, like the VASP [12]. However, it is difficult to scale the PWP method to very large number of processors. In the well known example of the Gordon Bell prize winning Q-box PWP code calculation, it has scaled up to 131,072 cores using MPI on the IBM Blue Gene/L machine [13, 14]. However, not every problem can be scaled to such numbers of cores. Such high scaling has been achieved either by using many k-points (e.g., 8 k-points) and large

† Corresponding Author
‡ 1 CPU/GPU computing unit = 1 CPU core + 1 GPU card
energy cutoff (112 Ryd), or large number of nonlocal projectors in reciprocal space (e.g., 64 nonlocal projectors per atom) and large number of electron states (e.g., 12000 electrons). As a result, the large matrix-matrix multiplication dominates the calculation. Recently, our own PETot PWP code has also been used to calculate 8,000 electronic states for a single k-point using 3,250 processors on the Jaguarfp machine in National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory [15]. Unfortunately, for many systems (e.g., semiconductor systems) with about one thousand atoms, there is no need to use many k-points, no need to use large number of nonlocal projectors per atom, and also no need to have ~10,000 electron states. Besides, it will be much faster to calculate the nonlocal pseudopotential projections in real space instead of in reciprocal space for large systems. This will reduce the size of matrix multiplications, thus further reduce the scalability. For many scientific problems, e.g., defect, alloy, liquid, surface, interface, grain boundary, and nanocrystals, a few thousand atoms could be large enough to describe the relevant physics. What is needed is a longer molecular dynamics (MD) simulation time, in other words, faster computations. Due to the inability to scale to larger number of processors, we cannot efficiently use the petascale computers for such tasks. Thus, there is an urgent need to speed up the DFT calculations for the few hundreds to a thousand atom systems in other ways. This will also help the calculations of much larger systems (with >>1000 atoms), since the divide-and-conquer O(N) scaling method can be a good way to calculate such large systems, as implemented in our Gordon Bell prize winning linear scaling three dimensional fragment (LS3DF) code [16], which is developed on top of the PETot code. In such a method, a large system is divided into many fragments which contain a few hundred atoms and will be calculated independently. Thus the speed up of such fragment calculation is critical to the overall speed of the LS3DF method.

One option to speed up the thousand atom DFT calculation is to use accelerators like the graphics processing unit (GPU). Surprisingly, despite of the recent race to implement the GPU computation in different applications, we have yet to find a work on DFT PWP calculations for GPU clusters in the literature. There are quantum chemistry calculations using GPU and the localized basis sets [17]. But the computational issues are very different from the DFT PWP calculations, e.g., the Fast Fourier Transformation (FFT) does not play an important role in such calculations. There is a work on GPU DFT calculation using wavelets (BigDFT) [18]. A speedup factor of 6 has been observed. However, the wavelet method is not the most widely used method for DFT calculations. There is a paper for PWP calculation using GPU, but just on a single GPU, and only the FFT has been accelerated using the GPU, and only a small 2 atom system has been tested [19]. The overall speedup for that work is rather modest: 15% decrease of the total computational time. In order to have bigger impact, what we need is an implementation which combines the MPI parallelization with the GPU acceleration for relatively large systems (e.g., up to a thousand atoms).

The optimal parallelization scheme using MPI for a DFT PWP calculation has been worked out 15 to 10 years ago [20]. This includes the data layout, the load balance, the FFT, the reciprocal (G)-space, band-index, and k-point parallelizations. Thus, for the dozen or so large scale DFT PWP codes [11], they all have similar parallelization schemes. Our PETot code [21] has implemented all the three levels of MPI parallelizations: G-space, band-index, and k-points. It can run ultrafast pseudopotentials as well as the norm conserving pseudopotentials. It includes three different algorithms for wave function improvements: the band-by-band conjugated gradient (CG), the all-band CG, and the residual minimization-direct inversion of iterative space (RMM-DIIS) method. It has also implemented different methods for charge density self-consistent iterations [22]. It has a special spherical G-space FFT [23]. Our PETot code can be used to calculate a thousand atoms system with about one thousand CPU processors. The best practice for GPU DFT-PWP calculation is still unknown. It is not clear what will be the best parallelization scheme combining MPI and the CUDA/GPU acceleration, and what is the speedup attainable under GPU. The situation is complicated by the fact that the multi-processor G-space sphere FFT is rather fragmented in terms of data distribution and computational task, and not easy to be accelerated by GPU. In this paper, we will test the GPU calculation using the DFT PWP method based on our fully functional CPU PETot code. We will emphasize on combining CPU MPI with GPU CUDA, and calculations on large (close to a thousand atoms) systems. Overall, we found a factor of 6 to 10 speedups for our GPU code compared with the original CPU code on the same machine. This is achieved mainly through a hybrid parallelization scheme: with both G-space and band-index parallelization, and carrying out FFT on a single GPU. We will use double precision calculation in the GPU, although single precision operation might further improve the performance. We will illustrate our data layout and parallelization schemes in detail, thus interested readers can follow our implementation.

2. ALGORITHMS AND CPU PARALLELIZATION SCHEMES FOR DFT CALCULATIONS

A basic flow chart of a DFT calculation is shown in Fig.1 [10].

![Figure 1. The flow chart of a DFT calculation. The DFT formula (e.g., LDA) is used to calculate V(r) from ρ(r). There are N electron wave functions Ψ, where 2N is the number of total valence electrons in the system. However, in actual calculations, the calculated total number of wave function N can be slightly larger than the number of occupied states. It involves the wave function improvement based on the Kohn-Sham equation HΨ = EΨ, depicted as the first box in Fig.1, and H is the single particle Hamiltonian. After the wave functions {Ψi} are solved, the charge density ρ(r) is calculated by occupying the wave functions. Then the output potential V(r) = V Coulomb(r) + V xc(ρ(r)) is calculated from ρ(r) using the Poisson solver for the Coulomb potential V Coulomb and the local density approximation (LDA) or general gradient approximation (GGA) formula for the exchange correlation potential V xc. Next, the output potential V(r) is compared with the input potential for self-consistency. If they do not agree, a potential mixing is used to generate a new potential for next self-consistent field (SCF)
iteration. In the PEtot code, Pulay pluses Kerker potential mixing are used to generate the next iteration potential, and they work well for most systems containing up to a thousand atoms. The most time consuming part of the whole calculation is to solve the wave functions according to $H\psi_i = \epsilon_i \psi_i$. For large systems containing a few hundreds to a thousand atoms, this step (the first box in Fig.1) can take about 99% of the total CPU computational time. Thus, in our GPU implementation, we will concentrate on this step.

There are many different algorithms to solve $H\psi_i = \epsilon_i \psi_i$, usually starting iteratively from a good initial set of wave functions. In the PEtot code, we have implemented band-by-band algorithm, all-band conjugated gradient (AB-CG) algorithm, and RMM-DIIS method. The AB-CG algorithm is stable, and relatively fast. Thus, in the current paper, we will concentrate on the AB-CG algorithm, which is illustrated in Fig.2. This algorithm starts with a subspace diagonalization (Sub-diag) among the N calculated wave functions. This time consuming step involves the calculation of the matrix $(i,j) = \langle \psi_i | H | \psi_j \rangle$, followed by the diagonalization of the matrix $h(i,j)$ (e.g., using the ScalAPACK pheev routine), then rotate the wave function $|\psi_i\rangle$ according to the eigen vectors of $h$. The AB-CG algorithm also ends with one Sub-diag step. In the middle, it uses a few (nline, e.g., from 2 to 4) conjugated gradient (CG) steps. The heavy computational steps in these CG steps include the $H\psi_i$ (to be called Hpsi), the projection of the residual vector $P_i$ from the $|\psi_i\rangle$ subspace (to be called Projection), and the orthogonalization among $|\psi_i\rangle$ (to be called Orth). We will call all the other parts (like the Precond CG step, the line minimization steps shown in Fig.2, and some other small parts, e.g., to normalize the state vectors) the Fortran-loops.

Here $G$'s are the reciprocal lattice of the real space periodic box. If we assume $A(3)$ are the three edge vectors of the periodic box, and $I(3)$ are their three inverse vectors, e.g. $I(1) = G(1)$, $I(2) = G(2)$, and $I(3) = G(3)$. Then

$$G(i_1, i_2, i_3) = 2\pi \left[I(1)i_1 + I(2)i_2 + I(3)i_3 \right]$$

and $i_1, i_2, i_3 \in [-\infty, \infty]$ are integers. One common parallelization scheme is to divide the G-space into individual columns along the first direction $n_1$, then distribute the columns among the $n$ nodes processors (which are depicted as different colors in Fig.3). This can guarantee approximately same numbers of G-points in different processors (thus a work load balance when the state overlap-matrix are calculated), and a load balance when the FFT is carried out. In this way of parallelization, MPI communications within the $n$ nodes are needed when Hpsi is calculated. This is first demonstrated in the parallel MPI-FFT routine, with its details described in Ref. [25]. The communication is also needed to calculate the overlap-matrix, $\sum |\psi_i \rangle \langle \psi_i |$, which is sphere centered at each atom with a radius cut off $r_c$ has also been cut into pieces and distributed among processors. Thus, the results of the dot product $\langle \phi | \psi_i \rangle$ need to be added up among the neighboring processors. This requires MPI communication. For the state implementation but with almost the same accuracy [25]. Thus, we will use a real-space implementation of this nonlocal potential projection. Given the above workflow for the AB-CG algorithm, we will concentrate on the AB-CG algorithm, which is also implemented in the PEtot code and then a rotation following the inverse matrix of $U$.

$$h(i,j) = \langle \psi_i | H | \psi_j \rangle$$

Sub-diag.

$$P_i = H\psi_i - \epsilon_i \psi_i$$

Hpsi.

$$P_i = A(P_i - \frac{\lambda_i}{\lambda_i} P^{(n)})$$

Precond. CG step

$$P_i = P_i - \sum_{j\neq i} \langle \psi_i | \psi_j \rangle$$

Projection.

$$\psi_i = \psi_i \cos \theta_i + P_i \sin \theta_i$$

Line minimiz.

$$h(i,j) = \langle \psi_i | H | \psi_j \rangle$$

Sub-diag.

The Orth step is done with a calculation of the overlapping matrix $S(i,j) = \langle \psi_i | \psi_j \rangle$, follow by a Cholesky decomposition $U$ of the matrix $S$, and then a rotation following the inverse matrix of $U$ [24]. The Hpsi step involves a FFT to transfer the wave function from the reciprocal $G$-space to a real space regular grid, multiply the real space wave function by a potential $V(r)$, then an inverse FFT back to the $G$-space. The third way of parallelization is the $G$-space parallelization. This is to distribute the $G$-space wave function coefficients $C(G,i)$ among the $n$ nodes MPI tasks. In a plane wave calculation, the wave function $\psi_i$ is expanded by plane wave basis set as (we will ignore the k-point in our notation for simplicity):

$$\psi_i(r) = \sum_{|G|<G_c} C(G,i) \exp(-i \cdot r \cdot G)$$

where $G$'s are the reciprocal lattice of the real space periodic box. If we assume $A(3)$ are the three edge vectors of the periodic box, and $I(3)$ are their three inverse vectors, e.g. $I(1) = G(1)$, $I(2) = G(2)$, and $I(3) = G(3)$. Then

$$C(G,i) = \sum_{n} \langle \phi_{n} | \psi_i \rangle$$

for other steps (Sub-diag, Projection, Orth), the state overlaps like $\langle \psi_i | H | \psi_j \rangle$, $\langle \psi_i | \psi_j \rangle$, $\langle \psi_i | \psi_j \rangle$ (we will call them collectively as overlap-matrix later in the text) need to be calculated. Note that it is the overlap-matrix calculation which makes the DFT calculations $O(N^3)$ scaling. These calculations will require communications between the processor groups. The wave functions can be passed to each group in a Round-robin style, or a transpose of the wave function $|\psi_i\rangle$ can be made to change the data distribution from band-index parallelization to a $G$-space parallelization as will be described now.

The time consuming steps are indicated by the asterisk signs. The other parts will be called collectively as the Fortran-loops. The scheme has been simplified in some ways for clarity.
overlaps matrix $\langle \psi_i | H | \psi_j \rangle$, $\langle \psi_i | \psi_j \rangle$, $\langle \psi_j | \psi_i \rangle$ (they are calculated in G-space), all the $(i,j)$ pairs can be calculated within one processor, but since only a part of G-space coefficients is located within one processor (Fig.3), a global sum (MPI_Allreduce) among the $n_{\text{nodes}}$ processors is needed to get the final results.

To summarize, there are different MPI communication requirements for band-index parallelization and for G-space parallelization. In the PEtot code, all the three types of parallelizations are used, thus the total number of MPI processors is $n_k \times n_{\text{nodes}}$, where $n_k$ is the number of processor groups for k-point parallelization. In the cases studied in this paper, $n_{k}=1$.

3. GPU HYBRID PARALLELIZATION SCHEME FOR DFT CALCULATIONS

As for the CPU+GPU (MPI+CUDA) code, we have tested different ways of parallelization. In the G-space $n_{\text{nodes}}$ parallelization ($n_{\text{group}}=1$) scheme, the FFT has already been broken into many 1D FFTs, it is difficult to use GPU to further accelerate the 1D FFT, and the MPI communication in the FFT overwhelms the floating point operation for relatively large $n_{\text{nodes}}$. We found that the GPU speedup is rather low in such direct G-space parallelization scheme. On the other hand, in the band-index parallelization scheme, the FFT can be done within one CPU/GPU, but the overlaps matrices are difficult to be calculated (large MPI data communications among the different CPU/GPU computing units).

One way to overcome this is to do a transpose of the wave function (when the overlaps matrices are to be calculated) from the G-space parallelization overall the CPU/GPU computing units, to a band-index parallelization. After this step, each CPU/GPU computing unit will do FFT within itself, and also do the nonlocal potential projection $\Sigma \langle \phi_l \rangle \langle \psi_i \rangle$ within itself for the $N/n_{\text{nodes}}$ wave functions $\Psi_i$ allocated to this node. Thus, there is no real space fragmentation as shown in Fig.4, and no MPI communication between different nodes. Avoiding the MPI communication for the $\langle \phi_l | \psi_i \rangle$ calculation is one big advantage of this scheme. After all the $\Psi_i$ have been calculated within each CPU/GPU computing unit, a MPI_Alltoall communication is used again to transpose the wave function back into the G-space parallelization (distribution) for the subsequent calculations.

One concern for the above G-space/band-index hybrid parallelization scheme is the memory requirement. The FFT is not distributed, thus at least one CPU/GPU computing unit needs to host an $n_k \times n_{\text{nodes}} \times n_{\text{group}}$ array. For a typical plane wave energy cut off Ecc (e.g., 30-50 Ryd) and the corresponding real space grid interval, a ~ thousand atoms system might require 128$^3$ to 256$^3$ grid points, depending on whether there are empty vacuum spaces. Thus, a double precision complex wave function in real space will only take 34 to 268 MB. Since the global memory for each GPU card (see Fig.6 later) can be from 2 GB to 4 GB, this is not really a problem for the systems we want to study. If there is no empty space, e.g., for crystal, alloy, liquid, it will thus be possible to calculate systems up to 10,000 atoms theoretically in this parallelization scheme. Another issue is the nonlocal potential projector $\langle \phi_i \rangle$ as shown in Fig.4. To execute the projection efficiently, all these projectors in real space should be stored in the GPU global memory. Again, for the typical real space grid interval used, and with a typical 1.8 Å radius cutoff $r_c$ for the spherical G-space FFT. As discussed above, the nonlocal potential projection is $\langle \phi_i \rangle$ (which are real) 5000 $\times$ 9 $\times$ 1000 $\times$ 8 = 360MB. This is still allowable for a 2 GB memory GPU card. The memory of the wave function $\langle \Psi_i \rangle$ depends on the number of CPU/GPU computing units used, and the number of valence electrons in each atom. For example, in our test cases below, for 32 processors, we found that the memory of the wave function is in the order of 30 MB (for the entire index “i” and G vectors). Thus, several copies of the wave functions can be stored in the GPU global memory. This is important as they are needed in the overlap matrix calculations, and the subsequent wave function rotations.

In the CPU code, in the G-space parallelization, a single FFT is done in a parallel way (although many FFTs can be stacked together for a block-FFT) as shown in Fig.3. A specially written FFT is used to take the advantage of the features that only the G coefficients inside the sphere shown in Fig.3 are nonzero [25]. Such special FFT can save the computational time by a factor of 2 to 3 compared with full box FFTs like the FFTW [25], or other standard library routines. In principle, the same can be done for the FFT in GPU. However, in practice, at this point it is too much effort to write a CUDA based spherical G-space FFT. We have thus chosen to use the full box 3D CUFFT library. Before doing the FFT, the G-points outside the sphere are patted with zeros, and then a full GPU FFT is carried out using the CUFFT library call. We found that such CUFFT is about 4 times faster than the CPU spherical G-space FFT. As discussed above, the $\langle \phi_l | \psi_i \rangle$ is also done in GPU after the FFT for each $\Psi_i$ (we do the FFT and the nonlocal potential projection in a band-by-band fashion in GPU, after the all-band MPI_Alltoall transpose of the wave functions).

One computational intensive operation is to calculate the overlaps matrix in Sub-diag, Projection and Orth steps shown in Fig.2. There are $N \times N$ elements for such matrices ($N \approx 1000$), and the

![Figure 3. The G-space data distribution in a parallel FFT.](image-url)
total number of G vectors (for all the nodes together) is about one million as will be discussed later. Since we still make the book-keeping in the CPU, every time when such overlaps are needed, the corresponding wave functions \(\Psi_i\) (for instance \(\Psi_i \equiv h_\Psi\)) will be copied from the CPU to the GPU (using cublas_set_matrix call). Then GPU matrix-matrix multiplication BLAS-3 routines (cublas_dgemm, cublas_zgemm) will be called to carry out the matrix-matrix multiplication. However, the resulting matrix h(i,j), S(i,j) etc (with a dimension of matrix-matrix multiplication. The obtained matrix h, S, etc 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_j), to be carried out again using cublas_dgemm or cublas_zgemm. The copying of the wave functions and the matrix h(i,j), S(i,j) between CPU and GPU does take some time, this is because the CPU-GPU connection bus is slow. Fortunately in our tests, we find that such time expense is tolerable. In the future, we plan to move the book-keep of the AB-CG algorithm to the GPU. That will save some of the copying times between CPU and GPU. Overall, the biggest saving in these overlaps matrix calculations comes from the use of cublas_dgemm and cublas_zgemm.

4. THE PHYSICAL TESTING SYSTEMS

We have chosen two systems to test our GPU PEtot code as shown in Fig.5. The first is a 512 atom GaAs bulk system with one As atom replaced by one N atom. This is an important system for potential solid state lighting applications as the N atom introduces an impurity state near the conduction band [26]. \(E_c=40\text{Ryd}\) is used, and the real space grid is \(128^3\). There are 1024 occupied electron states (in our calculation \(N=1100\)). The total number of plane waves (G-points in the sphere of Fig.3) is \(3.3 \times 10^5\). The second system is a 933 atom CdSe quantum dot (QD) passivated by hydrogen atoms placed in a large box. Such QD has many potential applications from biological tags to solar cells [27]. \(E_c=30\text{ Ryd}\) is used, and the real space grid is \(256^3\). There are 1416 occupied electron states (in our calculation, \(N=1500\)), and the total number of plane waves is \(1.1 \times 10^6\).

In both cases, norm conserving pseudopotentials are used, and the nonlocal potential projections are carried out in real space. For Ga, Cd, N and H atoms, each atom has 8 \(\phi_l\) projectors, while for As and Se atoms, each atom has 4 \(\phi_l\) projectors. The d-electron is not

5. THE GPU MACHINE

Figure 6 shows a schematic of the architecture of the GPU cluster Mole-8.5 in the Institute of Process Engineering (IPE), Chinese Academy of Sciences.

![Architecture schematics of one Mole-8.5 computing node. The 3GB is the GPU's global memory, while the 8 CPU cores (the circles) share the 48 GB main memory.](image-url)
paper, we will count one CPU core plus one GPU card as one CPU/GPU computing unit (this is not the computing node), and use it as our basic unit for parallel scaling. For each such unit, the CPU core + GPU card theoretical flops is 524Gflops in double precision. Note, one CPU core can only take control of one GPU. Thus, in the CPU/GPU calculation, we have only used 6 CPUs per computing node (with two other CPU cores idled). This is even true for our CPU PEtot calculation. This also means that we have not taken the advantage that the 8 CPUs share the 6 GPUs; instead, we use it as one CPU is connected only to one GPU. In the actual tests, we only have access to 4 computing nodes, not the whole machine. Thus, we have in total 258 CPU/GPU computing units. Note, we used double precision in our GPU calculations. Although single precision calculations are often tested in GPU, and might lead to higher speed-up, due to the uncertainties for their accuracies, we have not used single precision calculation in this work.

6. CUDA IMPLEMENTATION OF THE FFT AND NONLOCAL PROJECTOR KERNELS

In CUDA programming, maximizing parallel execution and optimizing memory usage are two main goals. In the CPU code, FFT and nonlocal potential projection operations can take 40% or more of the computation time. Thus, it is critical to speed up these two parts. In the execution, these two parts are close to each other, thus it is advantageous to consider them at the same time. In order to efficiently use the GPU, we have ported the FFT and nonlocal potential projection operations completely into GPU, thus there is no need to have GPU and CPU data copying during the operations. This however requires us to develop hand-coded CUDA codes.

As discussed before, our PEtot GPU code uses a hybrid G-space/band-index parallelization scheme, and we will do a full box FFT by calling the GPU CUFFT library. The bottleneck of using the CUFFT is the memory copying between CPU and GPU. One naïve way is to map the spherical G-space data C(G,i) [Fig.3] into the full FFT box and put the outside with zeros in CPU, then copy the full box data to GPU. In our tests on the IPE machine for the 128³ and 256³ grid, we found that such data copying will cost more than 50% of the total FFT time. We thus changed our strategy, instead of copying the full FFT box data, we only copying C(G,i) within the sphere shown in Fig.3, and do the sphere-to-box mapping in GPU. This reduces the data copying time between CPU and GPU by 80%, but will require us to write a CUDA based mapping code in GPU. In the mapping operations, multiple CUDA streams are used to increase the concurrency among different computing tasks (kernel). Using these techniques, the GPU mapping (from sphere to full box) is also much faster than the corresponding CPU mapping. After this mapping operation, the full box CUFFT is called, and a real space wave function \( \psi_i(r) \) is generated on the \([n_1,n_2,n_3]\) grid in GPU. Note that, we are doing this part (including the CPU to GPU C(G,i) copying) in a band-by-band fashion, so only one work array with the size of \([n_1,n_2,n_3]\) is needed in GPU. This is important because otherwise the GPU global memory might not be large enough.

After \( \psi_i(r) \) is obtained, the non-local potential projections \( \phi_r|\psi_i \) need to be calculated in real space. Note that, the \( \phi_r \)'s have already been stored in GPU global memory throughout the calculation (until the atom has been moved). This multiplication also needs to map the data index within the sphere (for \( \phi_r \)) shown in Fig.4 into the global index (for \( \psi_i(r) \)) of the \([n_1,n_2,n_3]\) array. One possibility is to do this mapping first, then do the dot-product by calling the CUDA cublas dgemm function. But we found that this is not efficient because there is only one wave function to do the multiplication (due to the band-by-band operations). We have thus written our own CUDA code for this operation. Note that the different atoms (the “l” index in \( \xi_l \)) can be carried out independently. This provides a chance to utilize GPU parallelization among the GPU cores. Another major advantage is that, the full \( \psi \) within the sphere shown in Fig.4 is within a single GPU, so no MPI communication is needed in this step, and no data copying between CPU and GPU is needed either. The whole operations are done within a single GPU. This is in contrast with the CPU code, where the real space sphere of \( \psi \) can be distributed among multiple CPU processors as shown in Fig.4. After the vectorM(l) = \( \{ \phi_r|\psi_i \} \) is obtained for each “l”, the projectors need to be written back to the wave function as \( \psi_i(r) = \psi_i(r) + \sum M_l|\phi_r \). As before, multiple GPU cores can be used to calculate different “l” simultaneously. However, there is a subtle issue: different \( \phi_r \) might write to the same r point in \( \psi_i(r) \) (i.e., there are overlaps of the spheres in Fig.4), thus care must be taken to avoid data over-writing (e.g., using temporary array and performing the final writing in sequential order, or to avoid the overlapping atoms to be written at the same time).

Our overall experience is that, first, by carefully managing the memory and taking advantage of GPU memory bandwidth, the index mapping could be done very efficiently; second, by moving more computation into GPU and maximizing the utilization of GPU cores, we can gain good performance in the nonlocal projection and FFT calculations. Furthermore, we did find that it is rather difficult to debug the MPI+CUDA code because the GPU data (during kernel operations) is difficult to be printed out.

7. THE TIMING RESULTS AND DISCUSSIONS

Note that the CPU runs are carried out on the same machine, but using only the CPU part of the machine (6 CPUs cores per computing nodes in Fig.6), while the GPU runs use both the CPU and the GPU parts, with each CPU core controls one GPU card. The GPU speedup over the CPU ranges from 6 times to almost 10 times.

This definitely demonstrates that the DFT PWP code can be accelerated by the GPU. Our speedup is similar to the speedups reported for many other ab initio applications. The GPU speedup for the wavelet based BigDFT is about 6 times [18]. The reported quantum chemistry GPU codes can have larger than 10 times speedup [17], but with a small s, p, angular momentum localized basis set, and with single precision GPU calculations. The GPU quantum Monte Carlo simulation can also have a speedup of 6 times [29] to 10-15 times [30], due to a faster determinant calculation. Our speedup is achieved by the reorganization of the data structure, and using a hybrid G-space/band-index parallelization scheme. It can take the advantage of the full CUFFT on a single GPU card, and the advantage of the overlaps matrix calculation with minimum communication. Although there are many previous reports on GPU calculations for quantum mechanical simulations as mentioned above, as far as we know, ours is the first to report the DFT PWP calculations on relatively large GPU clusters.

The total double precision floating point operation count for the 512-GaAs system is 5.15x10^{13}, and for the 933-CdSe system it is 3.32x10^{14}. These are measured from the Craypf performance monitoring package (which uses PAPI) on the NERSC Cray XT4
The parallel scaling of our GPU calculation in terms of the space nonlocal potential projection is used. This is lower than the typical 60% efficiency attainable when G-range from 21% to 43% from 256 CPU cores to 16 CPU cores. The efficiencies (using the CPU theoretical speed of 9 Gflos/CPU core) realize the full potential of the GPU flops, compared to the CPU and GPU codes. Our corresponding CPU code multiplications which can efficiently use BLAS-3 routines both in million to 5000). As a result, there are no large matrix-matrix multiplication (the dimension of 4 to 5, but also reduces the dimension of matrix-matrix projection. This decreases the overall time-to-solution by a factor of 2 to 5, but also reduces the dimension of matrix-matrix multiplication (the dimension of $\phi_i$ can be reduced from one million to 5000). As a result, there are no large matrix-matrix multiplications which can efficiently use BLAS-3 routines both in the CPU and GPU codes. The corresponding CPU code efficiencies (using the CPU theoretical speed of 9 Gflos/CPU core) range from 21% to 43% from 256 CPU cores to 16 GPU cores. This is lower than the typical 60% efficiency attainable when G-space nonlocal potential projection is used.

The parallel scaling of our GPU calculation in terms of the CPU/GPU computing units is shown in Fig.7. scaling of CPU

Table I. The computational time (second) and the overall GPU speed-up compared to CPU for runs with different CPU/GPU computing units counts and for the two test systems. Note only the AB-CG times in Fig.1 are accounted here. The other parts of the calculated in Fig.1 take less than 10% of the total GPU version computation time. The 933-CdSe system can only be calculated on 256 computing units due to memory limitation when N=1500 is used. However, when N=1200 (electron states) is used, it can be run on both 128 and 256 computing units, and their corresponding PEtot GPU times are 71 and 47 seconds, and CPU times are 579 and 348 seconds respectively.

<table>
<thead>
<tr>
<th>Computing units</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>systems</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>933-CdSe</td>
</tr>
<tr>
<td>PEtot (CPU)</td>
<td>842</td>
<td>450</td>
<td>255</td>
<td>152</td>
<td>104</td>
<td>495</td>
</tr>
<tr>
<td>PEtot (GPU)</td>
<td>87</td>
<td>49</td>
<td>27</td>
<td>23</td>
<td>17</td>
<td>56</td>
</tr>
<tr>
<td>Speed-up (PEtot)</td>
<td>9.7x</td>
<td>9.2x</td>
<td>9.4x</td>
<td>7x</td>
<td>6.1x</td>
<td>8.8x</td>
</tr>
<tr>
<td>Total flops (Tflops)</td>
<td>0.59</td>
<td>1.05</td>
<td>1.91</td>
<td>2.24</td>
<td>3.03</td>
<td>5.92</td>
</tr>
<tr>
<td>Efficiency</td>
<td>7.1%</td>
<td>6.3%</td>
<td>5.7%</td>
<td>3.3%</td>
<td>2.3%</td>
<td>4.4%</td>
</tr>
</tbody>
</table>

Table II. The GPU times (in seconds) and the speedups (SU) compared with the CPU code counter parts for different computational kernels illustrated in Fig.2.

<table>
<thead>
<tr>
<th>Computing units</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>systems</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>512-GaAs</td>
<td>933-CdSe</td>
</tr>
<tr>
<td>Hpsi</td>
<td>7.83</td>
<td>4.58</td>
<td>2.73</td>
<td>1.84</td>
<td>1.52</td>
<td>4.93</td>
</tr>
<tr>
<td>MPI_Alltoallx2</td>
<td>3.02</td>
<td>2.12</td>
<td>1.49</td>
<td>1.20</td>
<td>1.17</td>
<td>2.05</td>
</tr>
<tr>
<td>FFT</td>
<td>2.40</td>
<td>1.23</td>
<td>0.64</td>
<td>0.36</td>
<td>0.19</td>
<td>1.50</td>
</tr>
<tr>
<td>Nonlocal proj.</td>
<td>2.00</td>
<td>1.02</td>
<td>0.50</td>
<td>0.23</td>
<td>0.14</td>
<td>1.24</td>
</tr>
<tr>
<td>Hpsi SU</td>
<td>5.1x</td>
<td>4.7x</td>
<td>4.9x</td>
<td>4.0x</td>
<td>3.3x</td>
<td>6.2x</td>
</tr>
<tr>
<td>Sub diag</td>
<td>4.32</td>
<td>2.88</td>
<td>2.32</td>
<td>2.11</td>
<td>2.21</td>
<td>4.63</td>
</tr>
<tr>
<td>zheev</td>
<td>1.30</td>
<td>1.28</td>
<td>1.42</td>
<td>1.41</td>
<td>1.76</td>
<td>3.06</td>
</tr>
<tr>
<td>Sub diag SU</td>
<td>11.7x</td>
<td>9.3x</td>
<td>6.5x</td>
<td>5.5x</td>
<td>4.6x</td>
<td>8.3x</td>
</tr>
<tr>
<td>Projection</td>
<td>3.13</td>
<td>1.65</td>
<td>0.91</td>
<td>0.71</td>
<td>0.43</td>
<td>1.49</td>
</tr>
<tr>
<td>Projection SU</td>
<td>15.3x</td>
<td>14.6x</td>
<td>13.4x</td>
<td>8.8x</td>
<td>7.8x</td>
<td>12.7x</td>
</tr>
<tr>
<td>Orth.</td>
<td>4.81</td>
<td>2.67</td>
<td>1.5x</td>
<td>1.14</td>
<td>0.80</td>
<td>2.77</td>
</tr>
<tr>
<td>Zpotrf</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.62</td>
</tr>
<tr>
<td>Orth. SU</td>
<td>16.1x</td>
<td>15.6x</td>
<td>13.7x</td>
<td>12.9x</td>
<td>12.8x</td>
<td>16.4x</td>
</tr>
<tr>
<td>Fortran loop</td>
<td>2.15</td>
<td>1.13</td>
<td>0.63</td>
<td>0.38</td>
<td>0.25</td>
<td>0.87</td>
</tr>
</tbody>
</table>
of the serial LAPACK CPU Cholesky decomposition routine zpotrf. This routine takes about 0.25 second out of the 0.8 second in Orth for the 256 computing units 512-GaAs case. The use of ScaLAPACK routine pzpotrf takes almost the same time. The scaling of Sub_diag is even worse. In Sub_diag, besides the overlaps matrix calculation, it also includes the diagonalization of the matrix using the ScaLAPACK pzheev. This part takes about 1.4 second, it makes the Sub_diag more time consuming than Orth with a worse scaling.

Finally, the Fortran_loop part represents the parts in the CG algorithm (Fig.2) which remain to be executed as CPU Fortran do loops. It scales reasonably well with the number of nodes, which implies the MPI_Allreduce communication to sum the do-loop results from different nodes (e.g., to obtain the norm of a G-space vector) does not dominate the calculation of this part.

The scaling of the Hpsi is also shown in Fig.8, which does not scale very well. The Hpsi time includes the FFT, the nonlocal potential projection and two MPI_Alltoall calls to transpose the wave functions from G-space parallelization to band-index parallelization. Both the FFT, and nonlocal potential projection scale very well. This is because they are executed within each CPU/GPU computing unit independently with each other (larger number of computing units, smaller number of states within each node), and they are executed in a band-by-band fashion. From Table II, it is also interesting to note that the GPU FFT and GPU real space nonlocal projection take similar times. The FFT and the nonlocal projection are not the most expensive parts in the Hpsi. Instead, the non-scaling MPI_Alltoall takes a lot of time, and become dominant for large computing units cases in the 512-GaAs calculations. This affects the scaling of the Hpsi time. The MPI_Alltoall to transpose the wave functions is essential to the overall design of the algorithm, and there is no way around it. Its speed depends on the MPI library implementation and the bandwidth of the node-to-node interconnection. In our calculations, it is mostly limited by bandwidth, instead of latency. Note that, in any other DFT PWP implementations, such global communication cannot be avoided. For example, in the CPU FFT routine, there is an MPI_Alltoall communication with data amount twice as big as the sphere shown in Fig.3 [23]. When the other parts have been improved, the MPI_Alltoall becomes one of the bottlenecks. Overall, the GPU speedup for Hpsi ranges from 3 times to 6 times depending on the system size and the number of computing units.

Finally, the total times for each part of the kernels shown in Fig.2, and also some key parts of the calculations listed in Table II are shown in Fig.9. From Fig.9, we can see that for large computing units, the most expensive parts are MPI_Alltoall, pzheev, and Orth (mostly due to the Cholesky decomposition zpotrf). The dominance of pzheev and zpotrf is in agreement with a previous time analysis for what will happen when the DFT PWP calculation is to be significantly accelerated [31]. The scaling of the
ScalAPACK routines lags behind the scaling of the other parts [31]. In our case, perhaps some GPU CUDA version of the library (e.g., the CULA library [32]) can help to solve this problem. The problem of MPI_Alltoall is more fundamental. However, in our GPU machine, 8 CPU cores share the 48 MB memory. It is thus possible in the future to explore this feature of share memory (perhaps using MPI/OpenMP programming) to speed up the MPI_Alltoall communication in the wave function calculations step. Thus, only the computing node to computing node MPI communication is needed, not inside the computing node.

8. CONCLUSIONS AND FUTURE WORK

We have demonstrated that the GPU can be used efficiently for the DFT PWP calculations, and to speed up the computational time by a factor of 6 to 10 over the state-of-the-art plane wave DFT codes. These speedups are in-line with many other GPU quantum mechanical applications, including the wavelet BigDFT work [18], the quantum chemistry work [17] and quantum Monte Carlo work [29, 30]. But this is the first time large scale GPU DFT PWP calculations have been reported. The plane wave pseudopotential calculations have a very large user base. Thus applying GPU for such DFT PWP calculations will have a big impact in material science simulations.

A hybrid parallelization scheme between the G-space parallelization and band-index parallelization is essential for realizing our speedups. Most importantly, while the overlaps matrix calculations are carried out in G-space parallelization, the FFT and real space nonlocal potential projections are carried out in band-index parallelization. While there is no problem to simulate one thousand atom systems in this hybrid parallelization scheme, there could be an issue for memory limitation if the system is close to 10,000 atoms, mainly because the FFT cannot be hosted within one GPU, and the memory requirement for nonlocal potential projectors is too large. On the other hand, for such large systems, direct DFT calculation might not be the wise choice. O(N) schemes like the LS3DF method [16] can be used, where a large system is divided into fragments a few hundred atom a piece. The current AB-CG routine can be used to solve such fragments.

The GPU code can be scaled to a large number of CPU/GPU computing units, although its scaling is worse than the original CPU code. This is mainly due to a few remaining bottlenecks: the MPI_Alltoall communication to transpose the wave function over the nodes; the ScaLAPACK matrix diagonalization routine pzheev and LAPACK Cholesky decomposition routine zpotrf.

It will be important to further improve the speed of the calculations in the future. One way to do that is to improve its scaling, e.g., to make sure it can scale to a thousand CPU/GPU computing units efficiently. The MPI_Alltoall communication is difficult to be speed up significantly without improve the hardware interconnect; although the CPU share memory architecture (among the 8 CPU cores shown in Fig.6) can be taken advantage of in the future (e.g. using MPI/OpenMP). The pzheev and zpotrf LAPACK routines can potentially be speed up in the future with the GPU LAPACK library (e.g., the CULA [32]). Furthermore, one possible future work is to change the whole book-keeping for the AB-CG method from the current CPU calculation to GPU calculation. Not only this can reduce the Fortran-loop calculation time, it can also reduce the number of wave function movements between the CPU and GPU. That can further improve the node scaling. If these parts can be improved significantly, another parts are the MPI global sums (e.g., to get the overlaps matrix results), and the CPU-GPU memory connection speed. If the AB-CG can be speed up by another factor of 10, then the other parts in Fig.1 should also be improved. Currently, in our GPU calculation, the other parts take about 10% of the time, while AB-CG takes about 90%. However, to speed up the other parts should not be so difficult. For example, there are already reports for how to speed up the exchange correlation potential calculation $V_{xc}(\rho(r))$ under LDA or GGA using GPU [33]. Finally, the single precision calculation within GPU can be explored with careful tests and benchmarks for both reducing the size of the memory requirement, and the computing times.

Thus, overall, if a much better MPI_Alltoall and MPI global sum are available (through better interconnect, or special hardware), and pzheev and zpotrf can be significantly speed up by the CULA library, then it might be possible to speed up the DFT PWP GPU code by another factor of 10 by scaling the calculation to thousands of CPU/GPU computing units (the limit of CPU/GPU computing units is the number of electron state N in our current algorithm for band index parallelization). Then, each ab initio MD step might only take a few seconds for a thousand atom system (currently, it takes about a few minutes using the state-of-the-art CPU codes), and we will be close to the dream of simulating such systems for a few ns in a reasonable time, similar to what we are doing using classical force field methods right now.

9. ACKNOWLEDGMENTS

The numerical calculation is carried out on the GPU cluster Molecule-8.5 at Institute of Process Engineering, CAS. We’d like to thank our colleagues Peng Zhu, Zongyan Cao, Jiongming Zhang and Haifeng Lu for the technical support and providing all-night office during our testing. The work of the first 3 authors is supported by National Basic Research Program of China2010CB832702, NSF of China10972215, 60873113; 863 Program 2010AA012301 and 2010AA012402. The work of W. Gao is supported by NSF of China 11071047 and Science and Technology Commission of Shanghai Municipality 09ZR1401900. The work of L.W. Wang is supported by the Director, Office of Science, Office of Basic Energy Sciences, Materials Science and Engineering Division, of the U.S. Department of Energy (DOE) under Contract No. DE-AC02-05CH11231.

10. REFERENCES


There are many DFT PWP codes, here is a partial list: VASP, CASTEP, CPMD, ABINIT, PWSCF, PEtot, DACapo, SOCORRO, DFT++, PARATEC, DOD-PW, CP2K, SPHINX, and QBOX.

VASP: http://cms.mpi.univie.ac.at/vasp/


Wang, L.W., PEtot code: https://hpcrd.lbl.gov/~linwang/PEtot/PEtot.html
