A special purpose computer for ab initio molecular dynamics simulations

Lin-Wang Wang
Computational Research Division
Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA
e-mail address: lwwang@lbl.gov

I. INTRODUCTION

Density functional theory (DFT) [1] based ab initio simulation has become a major matured research tool for material science and chemical science. It amounts for about 75% of all the computer time in the material science [2] and chemical science categories [3] in major computer centers like the National Energy Research Scientific Computing Center (NERSC). It has also been widely used in biology and other scientific areas like environmental science and nuclear energy research. In recent years, due to the concern of computational energy efficiency, and the newly available designing tools for special computers, there is a revised interest in special purpose computers [4]. This is exemplified by the Anton machine [5] for classical molecular dynamics simulations for biological studies, and the proposed special machine based on energy efficient embedded processor technology for climate simulation [6]. In the field of physics, there is a long history of building special purpose machines for lattice quantum chromodynamics (QCD) simulations[7]. In this paper, we will discuss the possibility of building a special purpose machine for DFT simulations. The construction of such a machine will speed up the DFT molecular dynamics simulations by a thousand times, making it as cheap as current day classical force field simulations, enabling us to study many critical material science problems which are unfeasible using our current computers. Unlike some of the other special purpose machines which are built for a handful number of scientific problems, a DFT simulation machine can be used for a large class of problems. In a way, it is the rival of the Anton machine for quantum mechanical simulations. In this paper, we will discuss the computer specifications needed to construct such a machine, and the best computer architecture to realize our goal. We will see that by focusing on the DFT problem and the specific algorithms, the DFT machine can parallelize a fixed size problem much beyond what is possible for a general purpose machine [8]. We will also show that, on such a computer, some of the algorithms need to be changed accordingly. Overall, we believe it is possible to construct such a special purpose machine in the near future.

Our scientific goal is to perform a molecular dynamics (MD) simulation for 10ns within a week for a typical 1000 atom system. This goal is chosen due to its relevance to many critical material science problems. We expect high demands for such simulations based on the current trends in ab initio and classical MD simulations. In terms of the length scale, 10ns MD simulation can be used to study (among others things): simulated annealing, surface atomic structures and passivations, catalytic process, growth/synthesis steps, diffusion steps. All these are current day critical scientific problems, and many of them are poorly understood, awaiting for accurate numerical calculations. The 10ns MD can be used to get the local thermal dynamical fluctuations (free energies) (typically 10-50ps length will be enough for that purpose for homogeneous systems like gas or liquid), it is also long enough to perform some local geometry searches, and to find the minimum energy geometry close to the starting point. In some other cases, the direct 10ns MD might still not be long enough. But the corresponding computational capability (which is 10 million MD steps) can be combined with other techniques (e.g., transition path sampling [9], accelerated molecular dynamics [10]) to study such problems in a much longer time scale. Currently, these techniques are mostly used in classical force field simulations because the DFT calculation cannot provide sufficient computational steps to make these techniques useful. Our new machine can change this scenery. This machine will make the DFT calculation as cheap as the classical ones. Thus, many of the classical simulations done today can be carried out with ab initio accuracy and reliability. More importantly, many other problems where reliable force field parameters do no exist can also be studied by the ab initio simulations on this machine.

In terms of the system size, a thousand atom might not be very large, but it can capture many important physical features. It roughly corresponds to a 25 A size cubic. Since the atomic force (due to a local atomic displacement) decays faster than 25 A, this size scale can guarantee that the local chemistry will not be affected by the boundary situation at the other side of the system. Thus, this size should be large enough to study impurity, surface, catalysis, growth, and many other local chemistry and atomic structure problems. Besides, 1000 atom is the limit where direct DFT simulation algorithm might still be appropriate. For larger systems, O(N) scaling methods will be more efficient [11]. There are many O(N) methods, like the linear scaling three dimensional fragment method (LS3DF) we developed [12], which can be used to calculate much larger systems (e.g, 10,000 atoms) on the current general purpose machines. Here, we will focus on the 1000 atom systems, and try to make them run faster. Thus, this is a more challenging strong scaling problem.

Finally, we have requested that the calculation to be finished within one week. This is because we plan to use the whole machine in a dedicated mod. Thus, only one job can be run in a given time. In order for the machine to be shared by a community (e.g., with 50 users), and each user to use the machine for at least one project a year, each simulation must be finished within one week. This is an operation mode used in many other large user facilities, like the X-ray source from cyclotron radiation.

Currently, the fastest ab initio density functional code is VASP [13]. It can do 10 ps simulation within a week for a 1000 atom system using thousands of processors [14]. Thus, our goal is to push it one thousand times faster, to do 10 ns simulation within the same time. The parallelization of the VASP code reaches its limit by 1000 or 2000 processors for a 1000 atom system (more processors will make it slower). In this range of problem size, the rule of thumb for maximum number of processors to run VASP is one processor per atom. Our goal is to use concurrency beyond the one processor per atom limit, instead reach the limit of about 1 million processors for the 1000 atom problems. The reason for the feasibility of such large concurrency comes from the special design of the computer architecture, and a direct and careful mapping between the software algorithm and the hardware design. To reach our goal, the hardware and software need to be developed in a coherent way as will be discussed below.

In a MD simulation for systems with hydrogen, the typical time step is 1fs. Currently, VASP can perform a selfconsistent calculation (during the MD simulation) for a 1000 atom system within 1 minute [14]. As a result, it finishes a 10 ps simulation (10,000 steps) in about a week. In order to speed this up a thousand fold, we need to perform a self-consistent calculation in 0.06 second. This is a daunting task. It is in the time realm of classical simulations. In this work, we will focus on plane wave pseudopotential approach to solve the DFT equations [15]. Not only the plane wave approach is the most mature approach, we also believe that in this size regime, the plane wave approach could still be faster than, for example, the real space grid approach [16,17] or finite element approach [18] for the same accuracy. This is especially true if a well designed algorithm is used and mapped uniquely to the computer architecture so that the global communication in the FFT ceases to be a bottleneck. As will be shown below, in our analysis, we found that the bottleneck comes from the wave function orthogonalization and subspace diagonalization. These are not the problems of the plane wave methods. On the contrary, using plane wave basis will reduce the number of basis functions for the same accuracy (e.g., compared with real space grid method [17]). As a result, it will help the orthogonalization and subspace diagonalization steps. We will base many of our discussions on our own plane wave pseudopotential code PEtot [19] and the all-band conjugate gradient method we developed recently. PEtot can handle both norm conserving pseudopotential and ultrasoft pseudopotential. It now has three levels of parallelizations: k-points, reciprocal G-vectors, and band index. It has three different algorithms to solve the Schrodinger's equation: band-by-band conjugate gradient method, residual minimization method, and all-band conjugate gradient method. It can relax the atomic positions (much like in a molecular dynamics) using the ab initio atomic forces.

II. BASIC FORMALISM FOR PLANE WAVE DFT CALCULATIONS

In order to design the architecture which fit best for the algorithm, it is necessary to first describe the basic formalism of DFT [1], and the corresponding algorithm steps [15]. To calculate a system under density functional theory, especially under its local density approximation (LDA) [20], one needs to solve the minima of the total energy as a function of the single particle wave function $\{\psi_i\}$:

$$\begin{aligned} E[\psi] = \int \left[-\frac{1}{2}\sum_i \psi_i^*(r)\nabla^2\psi_i(r) + V_{ion}(r)\rho(r) \right] d^3r + \frac{1}{2}\int \frac{\rho(r)\rho(r')}{|r-r'|} d^3r d^3r' \\ + \int \rho(r)\varepsilon_{XC}(\rho(r)) d^3r + \sum_{R,R'} \frac{Z_R Z_{R'}}{|R-R'|} \end{aligned} \tag{1}$$

here the occupied charge density is

$$\rho(r) = \sum_i |\psi_i(r)|^2 \tag{2}$$

and $V_{ion}(r)$ is the ionic potential, and Z_R is the nuclei charge at R, and the function $\varepsilon_{XC}(x)$ is the LDA exchange correlation function. If there are 2M electrons, there will be M occupied states (wave functions) ψ_i (i=1,M) assuming that the system is not spin polarized, thus one wave function can be occupied with two electrons, one spin up, one spin down. The wave functions $\{\psi_i\}$ satisfy the following orthonormal condition:

$$\int \psi_i^*(r)\psi_j(r)d^3r = \delta_{i,j} \tag{3}$$

Finding the minimum of the energy in Eq.(1) is equivalent to finding the solution of the following Kohn-Sham equation (sometime it is also called Schrodinger's equation):

$$[-\frac{1}{2}\nabla^2 + V(r) + \hat{V}_{NL}]\psi_i(r) \equiv H\psi_i(r) = \varepsilon_i\psi_i(r) \tag{4}$$

here the total potential V(r) has the following expression:

$$V(r) = V_{ion}(r) + \int \frac{\rho(r')}{|r-r'|}d^3r' + \mu_{XC}(\rho(r)) \tag{5}$$

here the second term is the Coulomb potential due to charge density $\rho(r)$, and the third term is the LDA exchange correlation potential coming from the derivative of $\rho\varepsilon_{XC}(\rho)$. In Eq.(4), we have also introduced a nonlocal potential $\hat{V}_{NL}$, which is a nonlocal operator acting on the wave functions. This nonlocal potential is needed for pseudopotential calculations.

In the plane wave (PW) method, the wave functions (also called orbital) are expanded by plane wave basis set as:

$$\psi_i(r) = \sum_q C_i(q)e^{iq\bullet r} \tag{6}$$

Usually a periodic box (supercell) is chosen. Then the reciprocal lattice of the supercell defines a grid of q in the Fourier space. As a convention, all the q points within a sphere defined by a kinetic energy cutoff $E_c = \frac{1}{2}q_c^2$ are chosen in the summation of Eq.(6). In the PW method, the wave functions are kept in reciprocal space (q-space), represented by the coefficients $C_i(q)$. Major operations, like the enforcement of the orthonormal conditions of Eq.(3) are carried out in this reciprocal space representation. However, to carry out operations like the $V(r)\Psi_i(r)$, the wave function is transformed via FFT into the real space on a regular real space grid. After the V(r) and $\Psi_i(r)$ multiplication, the result is

Fourier transformed back to the reciprocal space. This dual space representation is illustrated in Fig.1.

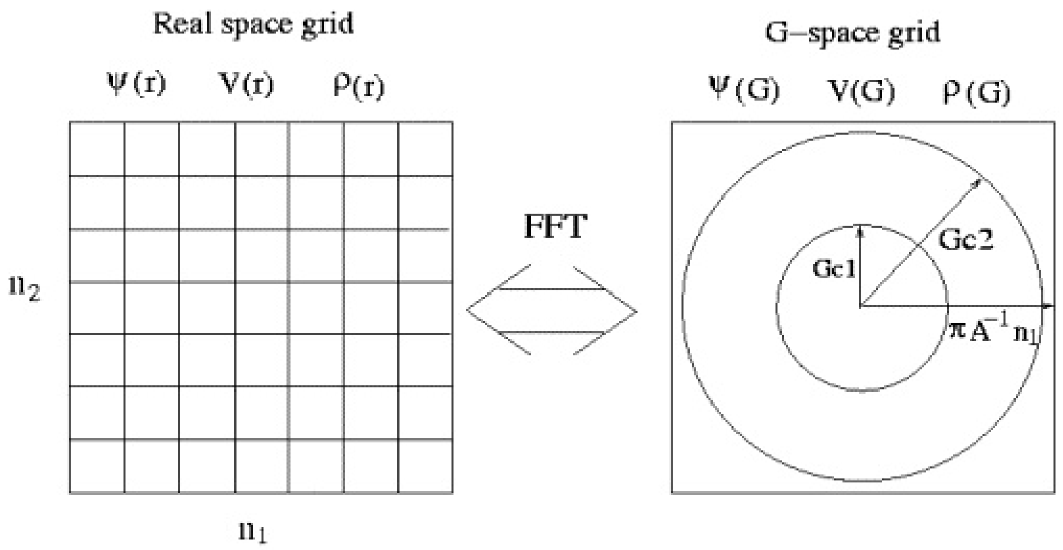

Fig.1, the dual space representation of the PW method. The reciprocal space (right box) and real space (left box). The wave function q vectors (G=q in this figure) inside the cutoff Gc1 are chosen in the summation of Eq.(6), while the potential V and charge density ρ are represented by the plane wave q inside a larger cutoff Gc2. Gc1 is half of Gc2. FFT is used to transform the wave functions from the reciprocal space representation onto the real space grid.

One computationally important term is the nonlocal potential in Eq(4). Using the Kleinman-Bylander approximation, the nonlocal potential is implemented as a reference function projection:

$$\hat{V}_{NL}\psi_i(r) = \sum_{R,l} \langle \phi_{R,l} \mid \psi_i \rangle \phi_{R,l}(r) \tag{7}$$

Here $\Phi_{R,l}$ are the atomic reference functions, and subscripts R, and l stands for atoms and angular moments. The dot product $\langle \phi_{R,l} \mid \psi_i \rangle$ can be evaluated either in reciprocal space or in real space. The $\Phi_{R,l}$ is localized in real space around each atom R. Thus, for large systems as discussed here, a real space evaluation can be much cheaper. Thus, we will use real space evaluation in the current paper.

A major computational task using the PW method for large scale parallel computer is the requirement for the FFT. While the number of floating point operation is moderate, the global communication for the FFT can be a serious bottleneck. Efficient FFT have been demonstrated using a few thousand processors in electronic structure calculations [21]. But for even larger number of processors, the communication message size can be very small, hence the communication latency can significantly slow down the FFT. This will be a major factor in determining the computer design and data layout for parallelization.

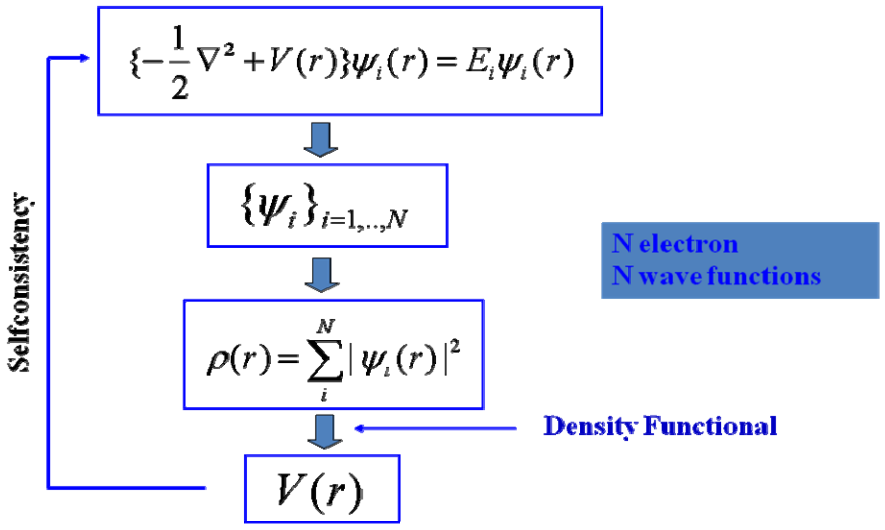

Fig.2. the flow chart of a DFT selfconsistent calculation.

Most modern DFT calculations follow a self-consistent charge mixing scheme [15] as shown in Fig.2. Under this iterative scheme, the wave functions $\{\psi_i\}$ are solved from the Schrodinger's equation [Eq.(4)] for a given input potential V(r), then the charge density ρ(r) is calculated from the wave functions, and a density functional (e.g., the local density approximation formula) is used to calculate the output potential V(r) [Eq.(5)]. This output potential is mixed with previous potentials to generate the input potential for the next self-consistent iteration [22]. The selfconsistency is reached if the input and output potentials are the same in this self-consistent loop. We will use N_{sc} to denote the number of self-consistent iterations. Note that, in practice, the Schrodinger's Eq.(4) needs not to be solved exactly at each outer self-consistent loop. Thus, the Schrodinger's equation is also solved using iterative methods. Although many iterative schemes exist, their computational costs can all be characterized by the number of $H\psi$ steps (iterations, denoted here as N_{iter}) and the number of times for the enforcement of the orthonormal condition [wave function orthogonalization, Eq(3)], and the number of orthogonalization projection between the new search direction P_i=H ψ_i and ψ_j [thus $\langle P_i|\psi_j\rangle$]. Usually, at the end of N_{iter} iterations, there will also be a subspace diagonalization among the wave functions, which requires the calculation of matrix $\langle\psi_{i|}|H|\psi_j\rangle$ and diagonalization of the resulting M x M matrix H(i,j).

In modern codes, for one MD step where good initial charge density and wave functions can be obtained from previous MD step, N_{sc} can be 5 to 10, while N_{iter} is about 2 to 5. One example is given in Fig.3, where a 1000 atom Si crystal with small random displacements is calculated with an all-band conjugate gradient method (as one of the algorithms to be used below). Here the system is initialized with random wave functions, and the self-consistent charge mixing scheme only starts from the 4th self-consistent field iteration (within N_{sc}, not N_{iter}). In the example shown in Fig.3, N_{iter}=5. In a MD step, since good initial wave functions and charge density can be obtained from previous step, usually the self-consistent charge mixing can start from the first iteration, and the total energy errors of the 5 to 10 N_{sc} steps are likely correspond to the 7th to 14th iterations shown in Fig.3. For a given N_{sc} and N_{iter}, there will be $N_{sc}*(N_{iter}+1)=10$ to 50 $H\psi$ (for each wave function) per MD step. Note, there will be $N_{iter}+1$ $H\psi$ for N_{iter} conjugate gradient steps. However, there is no need for further $H\psi$ for the subspace diagonalization.

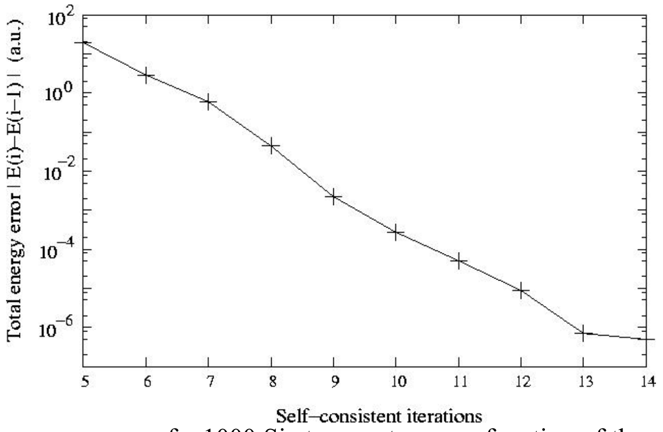

Fig.3, the total energy convergence of a 1000 Si atom system as a function of the self-consistent iterations. The convergence error is defined as the total energy difference between the current iteration and the previous iteration.

Table.I the major computational tasks, and the number of times they have to be executed for each molecular dynamic step within 0.06 seconds.

Task	Num	Note
$H\psi$	35	Equals Nsc*(Niter+1)
FFT	70	Within $H\psi$, 2FFT per $H\psi$, size $(200)^3$,
nonlocal	35	Within $H\psi$, Eq.(7).
P-orth-ψ	28 or 0	Equals Nsc*Niter or 0, dep. on the algorithm
Ψ-orth-ψ	7	Equals Nsc in our new algorithm
Sub diag $\langle \psi \mid H \mid \psi \rangle$	7	Equals Nsc in our new algorithm

In the following, we will use $N_{sc}=7$, $N_{iter}=4$, thus 35 $H\psi$ for our computer time analysis. This means that the time for each $H\psi$ step must be less than 0.06/35=0.0017 second. In Table.I, we listed the major computational tasks, and how many times needed to perform these tasks for each MD step (within 0.06 second). We have chosen the numbers for each task based on two possible algorithms. One is a new all-band conjugated gradient algorithm we developed recently [23], and implemented in our newest version of the PEtot code [19]. In this algorithm, there is no need to perform Gram-Schmidt orthogonalization for each conjugate gradient step. Instead, only one wave function orthogonalization step is needed for each self consistent step. Another algorithm is the residual minimization algorithm [24], which is implemented in PEtot code and the VASP code. This residual minimization algorithm is also called direct inversion of iterative subspace (DIIS) method in quantum chemistry [24]. Note that, if the residual minimization algorithm is used, there is no need for P-orth-ψ, thus its number is zero as shown in Table.I. In the following sections, we will design the computer architectures to reduce the computational time for each task listed in Table.I.

III. PARALLELIZATION SCHEME AND COMPUTER ARCHITECTURE

In our discussion, we will assume that there is only one k-point needed for our 1000 atom system. For a one k-point calculation, there could be two levels of parallelization based on the plane wave DFT formalism. If there are N_{tot} number of processors, they can be divided into N_g groups, each with N_P processors. Each group will calculate a subset of the wave functions (i.e, the number of wave function per group is $m=M/N_g$). This is called parallelization among the states (or say wave function index parallelization). Another level of the parallelization is the division of the q space (Gc1 sphere in Fig.1) by the Np processors within a group. This is done usually based on a column by column distribution to achieve load balance. This distribution is illustrated in Fig.4(a). Since the $V(r)\psi_i(r)$ will be carried out in real space, the real space grids also need to be distributed among the Np processors. This is done by dividing the real space grid into slices, and each slice belongs to one processor in Np. This is illustrated in Fig.4(b). The 3D FFT will be performed among the Np processors within each processor group.

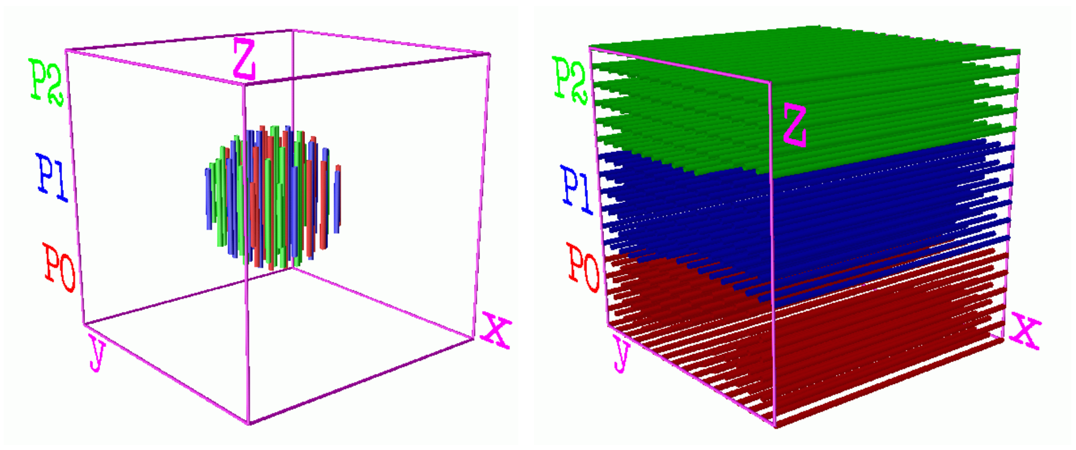

Fig.4 (a) the distribution of the q-space point to different processors (p0,p1,p2); (b) the distribution of real-space points to different processors.

To map the above two level parallelization in the algorithm into hardware, we have designed the following computer architecture, illustrated in Fig.5. In this architecture, the computer nodes will be divided into Ng groups, just like the wave function index parallelization in the algorithm. Within each group, there will be Nn nodes, each node will have Nc core. Thus, Np=Nn x Nc. The corresponding computer nodes in neighboring groups are connected by high speed data pipes (they will be called inter-group links). The connection within the group can be constructed in different ways, either as a 3D torus, or as a fat tree. A main task for the connection inside a group (in-group) is to perform the 3D FFT, and to have a fast global sum (all reduce) within the group. The FFT requires an all-to-all communication. In the next section, we will determine the parameters for the number of processors, interconnect speeds, the CPU speed, memory access rate and cache sizes, in order to achieve our goal of 0.06 second per MD step.

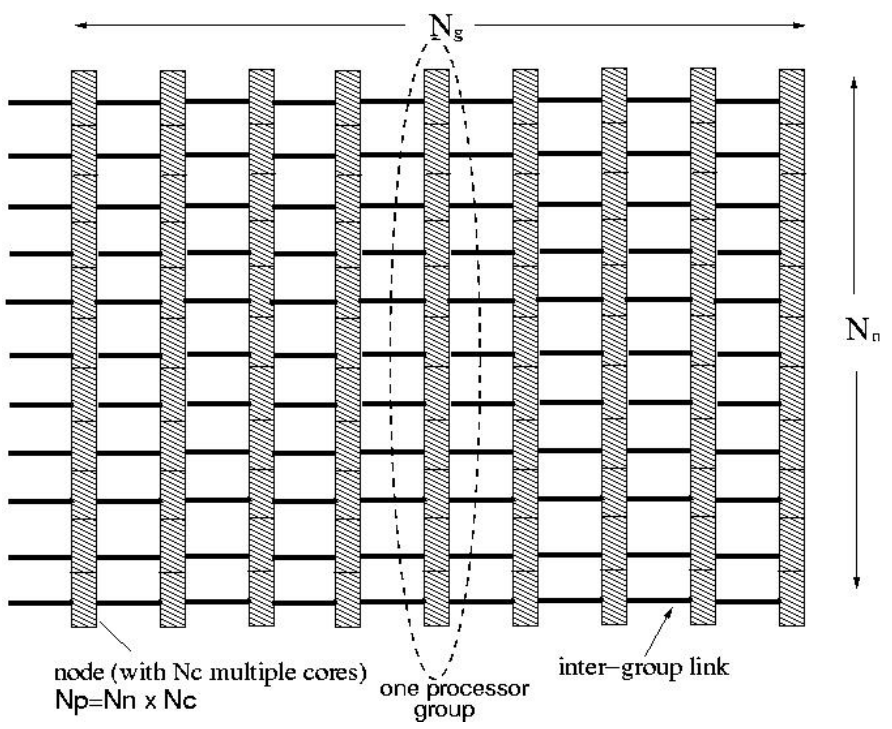

Fig.5, the computer architecture. Np is the number of processors within one group. Np=Nn x Nc, where Nn is the number of nodes within one group, and Nc is the number of cores within one node (socket). Ng is the number of processor groups. The connection between the corresponding nodes in neighboring group is denoted by a thick horizontal line, while the connection inside each group is not specified here. The links and communications between different processor groups are designated as inter-group, while the links and communications within a group are designated as in-group.

IV. **COMPUTATIONAL COSTS AND HARDWARE REQUIREENTS**

For a common semiconductor system, we will take the plane wave cut off Ec=35Ryd. If ultrasoft pseudopotential is used, this Ec can also be used for first row elements and transition metals. For most calculations, some kind of vacuum space exists, e.g., for a surface, or for a quantum dot calculation. Thus, we will assume a half space for vacuum, and half space for solid material. As a result, in average, there will be 1000 plane waves for each atom when Ec=35Ryd. For a 1000 atom system, this will also lead to a real space grid like n1 x n2 x n3=$(200)^3$ (assuming a cubic periodic supercell). If we are dealing with s-p atomic orbital systems, in average, each atom will have 4 electrons. That leads to 4000 electrons, thus M=2000 for the wave function index i. Note, if transition metal and d state is involved, this number can be increased by a factor of 3. For the rest of the paper, however, we will use M=2000. The total number of plane waves ng for each wave function is 1000x1000=1 million. Then the number of plane wave coefficients per processor for each wave function is ng_n=10^6/Np. From the past experience, we know that Np can easily be up to 256 for efficient FFT parallelization [21]. With better hardware parameters, we will push Np to be 1000.

Now, with Ng processor groups, we will have m=M/Ng wave functions per group. In the extreme case, we can have Ng=M, thus m=1, i.e, each group will deal only with one wave function. Actually, that is the parallel configuration recommended in VASP manual [25], especially when the algorithm of residual minimization is used. Now, if we also take Np=1000, then we have 2000 x 1000 =2 million processors. In terms of floating point operation, this will be a petascale computer, like the Roadrunner just released in the TOP 500 list [26]. Our computational parameters are summarized in Tabl.II. In the following, we will analysis the computational times for the major tasks listed in Table.I, and figure out the corresponding hardware requirement. As will be discussed below, each task in Table.I will involve two to three most time consuming major sub-tasks. This will amount for about 10 sub-tasks. Thus, in order to achieve the 0.06 second time for one MD step, we will require each sub-task to be finished within one tenth of the 0.06 second. This will also lead to a balanced algorithm and hardware design, which usually result in optimized the total performance.

Table.II, the basic computational parameters for a 1000 atom system and a 35Ryd plane wave cutoff. We have used 2000 processor groups each with 1000 processors.

Parameters	Value	Note
Ec	35 Ryd	Plane wave energy cut off
ng	1,000,000	Total number of PW per wavefunction
ng_n	1000	PW coeff. Per processor
nr_n	8000	Number of real space grid in each processor
Ng	2000	Number of processor group
Np	1000	Number of processors within each group
M	2000	Total number of wavefunctions
m=M/Ng	1	Number of wavefunctions per group
n1 x n2 x n3	$(200)^3$	FFT grid

(1) FFT calculations

Each $H\psi_i$ for one wave function i requires two FFTs, one forward, and one backward. Each FFT will be performed within one processor group, and there is one wave function for each group. Thus, assuming 70 FFT within each MD step as listed in Table.I, and assume FFT will only take one tenth of the total time, we have the time for each FFT: 0.06/70/10=0.0000857~0.0001 second. There are two major time consuming steps with one FFT. One is the data communication, another is the floating point operation. Each FFT requires at least one global transpose. Thus, the time for one transpose among the Np cores within one group must be smaller than 0.0001 second. Each wave function will have $ng=10^6$ complex coefficients, thus 16MB of data. The transpose is done on a cylinder after the first 1D fft (the 3D fft is broken down into 3 1D fft), which is three times of the q-space sphere shown in Fig.3. Thus, the total amount of data for the transpose for one wave function is 48MB. This will require a bidirectional (counting traffic in both direction) bisection bandwidth (among the group of Np processors) of 48MB/(0.0001 second)/2=240 GB/s. This is achievable if we form a 3D torus within the Np processors. With a 10x10x10 3D torus for the 1000 processors (assuming one core per node/scoket), the bisection bandwidth should be about 10x10x2xB_w=200B_w (factor of 2 for the bidirectional connection between two points in a torus), where B_w is a bandwidth for each link in the torus. Modern link can have B_w in a few GB/s, thus the bisection bandwidth should be achievable. This, however is assume that Nc=1 (i.e, one core per socket). If the number of multicore Nc per node is large, and the different core has to share the same link, the situation might change. But if B_w is about 5GB/s, there are still room for multicores, e.g, Nc can be 5. If Nc is very high, e.g, 10 or 100, then the number of node Nn will be small, an all-to-all connection among Nn should be used. In that case, the concern is not the bisection bandwidth, but individual node to node bandwidth. Such bandwidth should be: 48MB/Nn^2/(0.0001 second)=480/Nn^2 GB/s Thus, as long as Nn is larger than 10 (i.e, Nc is less than 100), this bandwidth is less than 5GB/s. Thus, with an all-to-all connection, this should be achievable. All these discussions are based on the assumption that the interconnects are between nodes (sockets), not directly between cores. In summary, for the requirement in FFT, if Nc < 10, we can have a 3D (or even higher dimension) torus connection. However, if Nc > 10, we should have an all-to-all connection among the Nn nodes. However, as we will see in later discussions, there will be stronger demand on interconnects (both inter-group and in-group) than on CPU speed. Since multi-core mostly increase the CPU speed, but not inter-connects speed (which are based on node/socket), we believe for our computer, it is better to keep Nc to be 1 or 2, unless the inter-connect speed (both latency and bandwidth) can increase proportionally with the number of cores. Thus, in many of our following discussions, we will implicitly assume Nc=1, thus Nn=Np. We will also assume a 3D torus interconnect within a processor group.

While the bandwidth might not be a problem, another concern is the latency time for the communication. Note, the average message size from one processor to another is only 48/Nn^2 MB. If Nn=1000, this is just 48 Byte, i.e, three complex numbers. But the communication time required is less than 0.0001 second, i.e, 100 μs. If the all to all communication is done in a serial fashion (e.g, like a isend-ireceive pair), and if Nn=1000, then the latency time for each pair need to be 0.1 μs. Current machines often have the latency time around one μs [27]. Then there could be a problem. However, the all-to-all communication can be implemented better than a serial of one-to-one communications. Thus, we believe 100 μs should be long enough. We also note that, the hard ware latency is in the order of 30 ns. The one μs latency in the current machine [27] comes mostly from the software layers. These layers are necessary to guarantee the reliability in a general purpose machine. But for a special purpose machine, and for a dedicated run for the whole machine, we believe the layers can be simplified significantly. It is reported that the communication latency between Anton processors is about 60 ns [5]. There, for classical simulations, the message size is also extremely small. Thus the latency is a major issue. The technology used in Anton can be applied in the current DFT machine.

As for the floating point operation for the FFT, each processor will get n1 x n2/Np=200x200/1000=40 columns. Each column perform a 1D fft for the length of 200 (which is about 3*n3*$\log_2$n3 FP operation, thus about 4000 operation). So, in total one processors needs to perform 40*4000*1.5=240,000FPO (1.5 is used because there are 3 1D FFT, but the numbers of other 2 1D FFTs are much smaller). To do this within 0.0001 second requires for 2.4 Gflops. This is just the normal speed of modern CPU. Thus, floating point operation should not be a problem for the FFT.

In actually FFT, we found that the data arrangement within each processor after the global data transpose might take significant time. This time is mostly caused by data access from the RAM. However, in our calculation, ng_n=1000, thus the wave function data is only about 16KB in reciprocal space. The real space data is nr_n=8000, thus 128KB. All these can be stored in the cache. Again, similar to the Anton machine, we can store the frequently needed data in the cache (if not the primary L1 cache, it will be stored in the secondary L2 cache) throughout the whole calculation. Note that, modern day computer (e.g, Cray XT4) has a 64KB L1 data cache, and 1MB L2 data cache [27]. For our computer, if we require L1 data cache to be about 300KB, then we can store two real space wave function in the cache, that should be enough for all our calculations, and ensure that all the wave functions stay in L1 data cache through out the whole calculation. This will be a major feature in our design, and it is a key step to reduce the communication time and data access time. For example, if all the data are in the cache, there is no difference between blas2 and blas3 libraries.

Overall, we found no intrinsic obstacle for the FFT calculation. This justifies our initial assumption that using plane wave basis is a good choice, and the FFT will not be a bottleneck in our calculation. Even in current computer and current code, for a thousand atom calculation, the FFT is not a computational bottleneck. In practice, we found that the float point operation (FPO) is not a problem, and the communication bandwidth should be fine if we have a 3D torus structure among the Np processors. We do need to be careful for how to implement the all-to-all communication for a given latency. Certainly, it should not be implemented in a serial isend/ireceive fashion. Finally, all the wave function data, both in real space and reciprocal space, should be stored in the cache, and remain in the cache through out the whole calculation.

(2) The nonlocal potential projection

As shown in Table.I, besides two FFT, the other heavy operation step for each Hψ operation is the nonlocal potential projection as described by Eq.(7). Assuming that the projectors $\Phi_{R,L}$ are all stored in local memory, then there is no communication needs among different groups of processors. Using a radius cut off Rc=3.2 Bohr for the real space sphere of nonzero $\Phi_{R,L}$ around each atom R, the number of grid point for each real space $\Phi_{R,L}$ is about 8000. Each atom will have 9 different L (1s+3p+5d states) if one reference state per angular momentum is used as in norm conserving pseudopotential, and 18 L if two reference states per angular momentum are used as in ultrasoft pseudopotential. For a conservative estimation, we will assume two reference states for each angular momentum in the following discussions. As a result, in total, the operation count for $\langle \phi_{R,l} | \psi_i \rangle$ for all {R,L} and one ψ_i is 8000*18*1000=1.44*10^8. We need to do this at least twice, one to get the dot-product, another to add the projectors back to the wave function as described in Eq.(7). Thus the total FPO count is about 3*10^8.

Each real space projector function $\Phi_{R,L}$ will be distributed among the Np processors within each processor group according to the portion of its real space sphere located at different nr_n real space division. For an averagely load balanced distribution (the load imbalance could be 50%), the FPO for each processor is $2*3*10^8$/Np=$6*10^5$ (the factor of two account for the possible 50% load imbalance). Now we need to finish this within 0.06/35/10=0.00017 second, thus the CPU flops for each processors is $6*10^5$/0.00017=3.5 Gflop. This is within the realm of current day CPU.

One issue however, is to perform a global sum. Since each $\Phi_{R,L}$ is distributed among different processors, the values for P(R,L)=$\langle \phi_{R,l} | \psi_i \rangle$ need to be summed up from its parts in different processors within a processor group. In the most brute force algorithm, the result for P(R,L) will be globally summed up from all the Np processors for all {R,L}. The data amount for P(R,L) is 18*1000*16=0.29MB. So, the question is: whether we can do a global sum of 0.29MB within the Np processors within 0.00017 second. To just send the 0.29MB data out from one processor within 0.00017 second, the bandwidth will need to be 1.7GB/s. To do an all-reduce global sum, it is likely that many times of such communications are needed. If we take a binary tree, that means $2*\log_2$Np =20 communication steps. Then the bandwidth for each interconnect will need to be 34GB/s. Although not impossible, this is about 10 times larger than the currently used interconnects. One possibility is to improve the brute force algorithm, because it is likely that one processor will not have all the atoms R. Thus, the relevant P(R,L) amount will be reduced, and the number of processors needed in the global sum for a given R will be smaller than Np. But very careful algorithm consideration is needed to take the advantage of these facts.

Another simpler way to solve the above problem is to store the whole sphere of each $\Phi_{R,L}$ for a given atom within one processor. Thus, one processor will take care of one atom, and no global sum is needed. This time we need to reassemble ψ_i from other processors for all its values within the sphere for a given processor. This is necessary, because after the wave function ψ_i is FFT from the reciprocal space to real space, the real space function is distributed evenly among the Np processors, and each processor has its own nr_n data point. The total amount of data to be sent (and to be sent back later) to a given processor is 8000*16=0.128MB (8000 is the number of grid point within each sphere). Thus, the require bandwidth is 0.128MB/0.00017s = 0.75GB/s. Thus, the bandwidth is not a problem in this regard. The latency should also not be a problem because probably only a dozen's other processors will send wave function data to this one processor, and the allowed total time is 170 μs. Note it is not efficient to do a standard global all-to-all operation for this purpose because each processor will not send data to all other processors. In summary, we believe that doing one atom by one processor is much cheaper than doing the atoms in a distributed fashion. Thus, in our requirement for the computer architecture, we will adopt this method. Nevertheless, the discussion of doing an in-group global sum in the above paragraph will still be useful, as it is pertinent to many other calculations to be discussed later.

Another issue for the nonlocal potential projection operation is its memory requirement. The memory requirement for $\Phi_{R,L}$ for each processor will be: 8000*18*8=1.2MB (here, the $\Phi_{R,L}$ can be made real, thus 8 Byte real number is used, this accounts for the factor of 8). If we want to store $\Phi_{R,L}$ inside the L2 cache, we need it to be larger than 1.2MB. We thus recommend 3MB L2 cache in order to store some other variables. This is not much larger than the L2 cache used in current computers (i.e., 1MB L2 cache in Cray XT4 [27]).

Finally, there is an issue of generating $\Phi_{R,L}$ when the atom moved after each MD step. We can distribute this task among the Ng groups. Thus, roughly one group will deal with one atom. To generate $\Phi_{R,L}$for one L, we need to do one FFT (from q-space to real space). Thus, we need to do 18 FFT for each Ng group. This is 70/18=4 times smaller than the FFT inside $H\psi_i$ within each MD step. Thus, this should not be a problem. After $\Phi_{R,L}$ is generated for one R within one processor group, we need to transfer it into all the other processor groups. This is very much like in the $\langle P_i | \psi_j \rangle$ orthogonalization or wave function orthogonalization tasks to be discussed below. The data amount to be transferred is 8000*18*8=1.2MB. This is to be compared with the wave function in G-space which is 16MB. Besides, the $\Phi_{R,L}$ only needs to be transferred once for each MD step, thus, if this is not a problem for the orthogonalization steps (there are 7 such steps within each MD step), it should not be a problem for $\Phi_{R,L}$ either. Finally, one might be able to develop algorithms to map $\Phi_{R,L}$ directly to the real space grid from radial and spherical Harmonics functions, without doing FFT. But more numeral tests are needed to establish the accuracy of such procedures for total energy calculations [28].

(3) $\langle P_i | \psi_j \rangle$ projection

Depending on the algorithm used, each $H\psi_i$ step might or might not need one $\langle P_i | \psi_j \rangle$ projection. If residue minimization is used, there is no need for this projection. On the other hand, if the all-band conjugate gradient method is used, each $H\psi_i$ step corresponds to one $\langle P_i | \psi_j \rangle$ projection. The conjugate gradient step is more robust, thus let's first assume this project is used, and then to show what is the corresponding architectural requirements.

In this step, $P_i=H\psi_i$, then we have

$$P_i' = P_i - \sum_{j=1,i} \langle P_i | \psi_j \rangle \psi_j \qquad (8)$$

The challenge of this step is that each P_i needs to be projected for the wave functions of all j (from 1 to i). For the purpose of estimating the computing time, we can replace j=1,i by j=1,M, thus treat all the wave functions in an equal footing. This step is the most computational and communicational demanding step. It is the most likely bottleneck of the whole calculation.

One major task to carry out Eq(8) is the data communication between different processor groups. This is because P_i is located at each processor group, while it needs to have a dot-product with all the wave functions $\{\psi_j\}$which are located in other processor groups as shown in Fig.5. In order to carry out this operation, we can pass each ψ_j from one processor group to next neighboring processor group (e.g., from left to right) in a round-robin fashion. After Ng such steps (hops), one ψ_j will have traveled through all the processor groups, and allow the $\langle P_i | \psi_j \rangle$ to be carried out for all the P_i in different processor groups. In order to finish this within the time of each $H\psi_i$ (which is 0.00017 second), the time to pass one wave function from one group to its neighboring group should be: 0.00017 second/Ng ~ 0.1 μs. The corresponding requirement for bandwidth of each inter-group interconnect in Fig.5 is: Nc*16KB/0.1 μs=160*Nc GB/s, here the Nc is the number of cores within each node. This is certainly the most stringent requirement. Even for Nc=1, the 160GB/s bandwidth interconnect is more than 10 times larger than what is deployed in current computers [27]. Note that, this bandwidth requirement cannot be reduced by reducing the number of processor groups Ng. When Ng is smaller, there are less steps for each wave function to hop, but for each step, there are more (m=M/Ng > 1) wave functions to be transferred. Also notice the advantage of using plane wave as the basis, instead of using real space grid (where the number of wave function coefficient can be 5 times larger for the same accuracy). Otherwise, the requirement for this bandwidth can be 5 times larger.

One possible solution for the above problem is to relax the requirement for asking this step to take $1/10^{th}$ of the total computational time. Instead, recognizing the large communication and computational demand, we can ask this step to take ½ of the total computational time. This will reduce the bandwidth requirement to 32 GB/s for Nc=1. This requirement is inline with the other requirements to be posed later.

For the latency, as stated above, each communication has to be done within 0.1 μs. Thus, the latency needs to be less than 0.1 μs. This should be achievable as we discussed above. The shortest latency demonstrated by the Anton machine is about 60 ns [5]. The key is to reduce the communication layers used in general purpose multi-user communication protocols. If ½ computing time requirement is used, then the latency time can be increased to 0.5 μs.

For the FPO, the number of multiplication on each processor for one wave function hop (within 0.1 μs) is 2*ng_n=2000 (two, for both to get the dot-product, and then to calculate the projection). Thus the FPO is 2000/(0.1 μs) = 20 Gflops. This is 4 times larger than the modern CPU. Since the CPU speed is not likely to be increased significantly, one possibility is to increase the number of cores within one node, but keep the number of Nn to be 1000 (thus to increase Np beyond 1000, and the total number of core beyond 2 million). On the other hand, as we discussed before, if we allow this step to take half of the computing time, then the flops requirement will reduce to 4 Gflop, which is within the realm of current day CPU. Also notice that, the floating point operation and the wave function communication can be overlapped to save time.

There is another global sum issue within each processor group. In order to get the dot-product $\langle P_i | \psi_j \rangle$, we need to perform an all-reduce global sum with the Np processors in each processor group. This is just one complex number, thus the bandwidth should not be a problem. But it has to be finished within 0.1 μs (if we require $p(j) = \langle P_i | \psi_j \rangle$ to be calculated for each j while ψ_j is still there). This is almost an impossible request, because if we use the 20 communication steps analysis of Sec.IV(2) for a global sum with the Np processors, that will require the interconnect latency to be 5 ns. This is far less than anything available now, and it is probably impossible to achieve any time soon. However, one can wait for p(j) for all $\{\psi_j\}$ have passed, then to do a global sum of p(j) for all the j at the same time. This time, the time allowed for the global sum is 200 μs, thus the latency is not an issue. Again, following the argument of Sec.IV(2), since the data size for the global sum is only 32KB, that will lead to an interconnect bandwidth requirement of 3.8GB/s, which is within the realm of current interconnect. Again, if we allow this step to take half of the total computing time, the all the relevant request will be 5 times smaller. One result of doing the global sum of p(j) for all j is that, the wave function ψ_j either need to be passed again when Eq.(8) is calculated, or they need to be stored in the memory, where a large memory access rate will be needed, see the discussion in the next section.

Overall, as we will see from the following discussions, the $\langle P_i | \psi_j \rangle$ projection step is the most demanding step. Its computations demand is similar to the wave function orthogonalization and subspace diagonalization as to be discussed below, however, each of the $\langle P_i | \psi_j \rangle$ need to be calculated in 1/4th of the time as requested for the orthogonalization and diagonalization steps. This is because one $\langle P_i | \psi_j \rangle$ needs to be done for each $H\psi_i$, where the orthogonalization and subspace diagonalization need only to done once for every N_{iter} (=4) $H\psi_i$. There are two solutions for this problem. One is to allow this step to take ½ of the whole computational time (since it is the most time consuming step) in the all-band conjugate gradient method, or we will use the residual minimization algorithm for most of the calculations (where the $\langle P_i | \psi_j \rangle$ step is not used). In reality, we might use a combination of these strategies, since the all-band conjugate gradient method is more stable, but the residual minimization method can be twice as fast due to the lack of this $\langle P_i | \psi_j \rangle$ step.

(4) Wave function orthogonalization

The next task in Table.I is wave function orthogonalization among $\{\psi_i\}$. Although the $\langle P_i | \psi_j \rangle$ projection step can be eliminated if we chose the residual minimization algorithm, the wave function orthogonalization step will always be there no matter what algorithm do we chose. This should be a Gram-Schmidt orthogonalization scheme. For the two algorithms we considered here, either the all-band conjugate gradient method, or the residual minimization method, there need to do only one orthogonalization for each self-consistent step. Thus, within each MD step, in average, we only need to do about 7 orthogonalizations. This is about 5 times less than the $\langle P_i | \psi_j \rangle$ projection step.

In both of the our algorithms, the fastest way to carry out the Gram-Schmidt orthogonalization is first to calculate the overlapping matrix: $S(i,j) = \langle \psi_i \mid \psi_j \rangle$, then do a Cholesky decomposition for matrix S: S=U^H*U. Here, U is an upper triangle matrix. Now, we can invert U to get U^{-1}, so that U*U^{-1}=I. Note, U^{-1} is still an upper triangle matrix. Now, we have $(U^{-1})^H$*S*U^{-1}=I. The new wave functions can be calculated as:

$$\psi_i' = \sum_{j \le i} U^{-1}(j,i)\psi_j \qquad (9)$$

It is easy to show: $\langle \psi_i \mid \psi_j \rangle = \delta_{i,j}$, and since in Eq.(9), the summation is only done for j less or equal to i, thus this is a Gram-Schmidt orthogonalization procedure.

The main computational parts to carry out this orthogonalization procedure include the calculation of the matrix S, the Cholesky decomposition of S to get U^{-1}, and carrying out Eq.(9).

The main cost to calculate S matrix is to transfer $\{\psi_i\}$ crossing the processor groups, much like in the $\langle P_i \mid \psi_j \rangle$ projection step. In our current layout, each processor group has only on $\psi_{i..}$ Again, we will do a round-robin style communication for $\{\psi_i\}$. Each ψ_i will pass from one processor group to its neighboring group in one hop. There will be Ng hops to finish the round-robin loop. Thus, the time allowed for each hop is (using the 1/10th of the computing time rule): 0.06/7/10/Ng=0.43 μs. The bandwidth requirement for the inter-group interconnect link is: 16KB/0.43 μs=37 GB/s. Again, this is about a factor of 5 larger than the currently used interconnect links [27]. But this might be possible in the next generation interconnect.

The latency for the above communication is 0.43 μs. Although this is shorter than the communication latency in most current day computers, but as we discussed above, this should be possible as demonstrated by the Anton computer.

There is also some communication issues for S(i,j). First, there is a need for a global sum within the Np processors within a group (in-group) for a given row of S(i,j) (i.e, a fixed i). This is much like the global sum issue for p(j) for $\langle P_i \mid \psi_j \rangle$. Here, the data amount is 16M=32KB. As for p(j), to reduce the latency, this can be done after all the ψ_j has passed through the group. Thus, we need to do this within 0.43 μs*Ng=860 μs. The demand for the in-group global sum can be referred to the global sum discussion in Sec.IV(2) for P(R,L) in nonlocal potential calculation. But here, the data amount is 9 times smaller, and the allowed time is 5 times longer. As a result, the in-group interconnect link bandwidth requirement is only 0.75GB/s, thus it is not a problem. After this global sum operation, each processor within a processor group will have the whole row of S(i,j) for a fixed i. In order to perform the Cholesky decomposition, we need to pass all the other rows from all the other groups to any given group. This can be done in the same way as we pass the wave functions $\{\psi_i\}$ through the processor groups. Now, the data amount on each processor is one row of S(i,j), thus 16*Ng=32KB. This is twice as much as the wave functions, thus the bandwidth requirement for the inter-group interconnect link is 32KB/0.43 μs=74 GB/s. This is quite large. Another option is to only transfer part of the row from each processor (since all the processors within one group have the same row of S(i,j)), but different processors transfer different part of the row. Thus, the bandwidth requirement for the inter-group interconnects can be reduced. Then the task will be shifted to a broadcast within the group after different parts of the row have arrived in different processors within a group. It is not clear whether this will be faster than the transfer of the whole row from each processor. However, the best strategy is to realize that, later on we will use scalapack to do the Chelosky decomposition with Np processors within the group. The scalapack requires that the matrix to be divided into a $\sqrt{N_p} \times \sqrt{N_P} = 32 \times 32$ grid, with each processor having one block of the matrix for its corresponding grid point. Thus, it is not necessary to ask each processor to have the full matrix. As a result, for a given processor (and all the corresponding processors from the other groups on a given horizontal line of Fig.5), the data amount needed to be transported for each hop is not a full row of the matrix, instead, it is only 1/32 of a row for its corresponding block. As a result, the corresponding inter-group interconnect bandwidth requirement is only 2.3 GB/s, well within the current day capability, and this requirement is much smaller than the requirement from wave function transportation. Note that, if only a block of the matrix is stored inside each processor, the data size for S is only M/32 x M/32 x 16 = 64KB. Thus, it can be stored inside the L1 cache, which will make all the calculations and communications faster.

After the S is decomposed and U^{-1} is found, Eq.(9) need to be carried out. At this stage, $\{\psi_i\}$ will be needed again at each processor group. There are two approaches. One is to do another round-robin communication, another is to store $\{\psi_i\}$ from the previous communication loop when S was calculated. The total data size for all the wave functions at a given processor is 16*ng_n*M=32MB. This cannot be stored in the cache. Thus, has to be stored in the RAM (another option is to request a L3 cache with 32MB). The floating point operation to calculate S and to carry out Eq.(9) is: 2*ng_n*M/(0.06s/7/10) = 4.6 Gflops (the factor of two: one for calculating S, another for calculating Eq.(9) for the wave function ψ_i' in this processor group). This is within the range of modern day CPU. When calculating Eq.(9), one challenge is to access the wave function data ψ_i from the memory (if we store the wave function from the first round-robin loop into the memory). Eq.(9) is essentially a matrix-vector multiplication (for each processor group "i"). Thus, in order to keep up the floating point operation of the CPU, we require a CPU to memory access rate of 16*ng_n*M/(0.06s/7/10)=37 GB/s. This is basically the same requirement as we have for the bandwidth for the inter-group interconnect. Now, the requirement is for the CPU to memory access. Current day memory access bandwidth is typically around 10 GB/s [27]. Thus, the required memory access might become possible in next generation machine. Besides, if such memory access is not available, but the high speed inter-connect link is available, then we can carry out Eq.(9) by transferring ψ_i between processor groups, instead of getting them from the memory. This is the same procedure as for calculating the S matrix. One thing we can learn here is that, if the memory is going to be used to store large data, then the memory access speed should be similar to the inter-connect speed in order to have a balanced machine.

So far, we haven't discussed how to decompose the 2000x2000 S matrix within the time of 0.06/7/10=0.001 second. One conventional way is to use Scalapack to do this Cholesky decomposition. This will be done within one processor group. Although this results in 2000 time repetition (all the processor groups are doing the same thing), but we believe it is difficult to use more than 1000 processors for a Scalapack application for a 2000x2000 matrix. According to Scalapack manual [29], the time to run a Scalapack routine can be estimated as:

$$T(N,P) = \frac{C_f N^3}{P} t_f + \frac{C_V N^2}{\sqrt{P}} t_v + \frac{C_m N}{N_B} t_m \qquad (10)$$

Here N is the size of the matrix. For us, N=2000. P is the number of processors, N_B is the size of the communication block. Usually it is assumed that the P processors divide the NxN matrix on a grid m x n, then the size of each grid is N_B. Thus, if we assume a square division, we have $N_B = N/\sqrt{P}$. t_f in Eq.(10) is the CPU time for one floating point operation. t_v is the time to communicate one item (i.e, a complex number here), this is determined by the bandwidth, and t_m is the communication latency time. In the formula, it is not specified what is the topology of the processor connections, and it does not distinguish the communication time and latency time from one processor to its neighboring processor, or to far away processors. It is assumed that there is a homogeneous connection from one processor to any other processors, and the times t_v and t_m are the times from one processor to any other processors in the P processor group. Thus, let's use our upper limit of our requirements discussed above to estimate T(N,P). Let's assume a 5 Gflops CPU, thus $t_f=2*10^{-10}$ second. Let's assume the in-group inter-connection bandwidth is 37GB/s, thus $t_v=4.4*10^{-10}$ second. Finally, let's assume the latency t_m to be 0.1 μs. The coefficients C_f, C_v, and C_m depend on the subroutines. We can use the PxPOSV scalapack routine for Cholesky decomposition. The corresponding coefficients are: $C_f=1/3$, $C_v=2+0.5\log_2 P$, and $C_m=4+\log_2 P$. Plug these parameters into Eq.(10), if we assume that P=1000 (i.e., using all the processors in one group for this task), then $N_B=63$, we have: $T(N,P)=(5.3+3.8+0.4)*10^{-4}$ =0.00095 second. This is just within the 0.001 second limit. There is a question about whether when P=1000, the estimation in Eq.(10) is still valid. Judged by the data block size $N_B=63$, we believe the estimation formula should still be good, especially as we discussed above, all the matrix can be stored in L1 cache. Although actual tests might be necessary to confirm the above conclusion, especially the dependence on the connection topology within Np, and the possible difference for the bandwidth and latency between neighboring processors and far away processors (if we use a 3D torus structure within each group), our estimation indicate that it is feasible to use scalapack to do the Cholesky decomposition. In the above estimate, we have not included the time to invert U. However, inverting a triangle matrix U should take less time than Cholesky decomposition itself. Thus, we believe this should not be a problem.

There is another possible way to find U^{-1}. Using our current LDA PEtot code, and the 1000 atom Si example shown in Fig.3, we found that the 2000x2000 S matrix is very close to I in almost all the self-consistent iterations. Thus, it is possible to find U^{-1} by using expansion serial. Basically, for the first order, $U^{-1}(i,j)=2\delta_{i,j}-S(i,j)$. Then, one can calculate $S'=(U^{-1})^H*S*U^{-1}$. We found that in S' is usually much closer to I than S. This procedure can be repeated for several times (i.e, doing the same thing for S'). Eventually the resulting matrix is I. Then, the final $U^{-1}(\text{final})=U^{-1}(1)*U^{-1}(2)\ldots$ (which is still an upper triangle matrix). Unless the first perturbation formula is already good enough, this procedure however requires matrix-matrix multiplications, which is N^3 operations. Since the Cholesky decomposition in lapack and Scalapack requires only $N^3/3$ operations, looks like this way will be slower, unless the first order perturbation is already good enough, or the whole machine (for all the processor groups) can be used for the matrix-matrix multiplication. But then the problem shifts to data communication between the processor groups. Thus, we will chose to use the Scalapack routine for the Cholesky decomposition step.

(5) Subspace diagonalization

Our last major task listed in Table.I is the subspace diagonalization. This involves the calculation of $H(i,j)=\langle\psi_i \mid H \mid \psi_j\rangle$ matrix, then diagonalize this matrix and find out the eigen vectors $V_i(j)$, and construct the new wave function as

$$\psi_i' = \sum_j V_i(j)\psi_j \qquad (11)$$

Usually, the subspace diagonalization step is done after the wave function orthogonalization step. One $H\psi_i$ is carried out for the wave function ψ_i in its own processor group, and then the dot-product is calculated when other wave functions ψ_j passing through this processor group in the round-robin loop.

The passing of the wave functions from different processor groups through each processor groups in the round-robin style, the calculation of dot-product, the in-group global sum to get H(i,j) (for a given i), and the calculation of Eq.(11) are all the same as in the wave function orthogonalization step. Thus, the corresponding architecture requirement in that step is also the architecture requirement in this step.

The issue for how to transport the H(i,j) from other processor groups (for different i's) to a given processor group might be different from that of S(i,j). If Scalapack will be used as for S, then the data layout (one block of the matrix for each processor) will be the same, hence the transporting procedure and requirement will be the same. But as we will find out below, we will probably use an iterative procedure to diagonalize H. For that purpose, each processor within a processor group will only need to have two column (or two rows, they are the same due to the symmetry) of the matrix. Since different processor groups have different rows, this results in a transporting of only two complex numbers within each round-robin hop. Thus the corresponding requirement for inter-group interconnect bandwidth is much smaller than that of S.

The more important difference than the wave function orthogonalization is the diagonalization of matrix H(i,j). The computational cost for diagonalizing a matrix is much higher than Cholesky decomposition. Again, let's assume to use the Scalapack to diagonalize the matrix. Eq.(10) will be used to estimate the time. We will use the routine PxSYEV for the diagonalization. In this case, the coefficients in Eq.(10) are: C_f=22/3, C_v=5log$_2$P, C_m=17/2*N_B+2. Again, we will use N=2000, P=1000, NB=63. Plug in Eq.(10), we have: T(N,P)= (117+28+17)*10^{-4} = 0.0162 second. This is about 15 times larger than the 0.001 second we set for each step. As we see from the formula, the most time comes from the floating point operation (the first term in the formula). Since we don't expect the CPU speed to increase ten times in the near future, this matrix diagonalization step might pose a problem.

However, the above estimation is based on the fact that all the processor groups will work on the same matrix independently. Thus, there is a tremendous waste of computing resources due to repetition. On the other hand, it is hard to imagine to use more than 1000 processors to work on an 2000x2000 matrix in the Scalapack routine. In current computers, the experience is that the Scalapack will stop to scale up after about 200-400 processors for such sized matrix. Assuming faster interconnect (as in our proposed computer), and faster data access (we store the matrix in the cache), it might be possible to extend this to 1000 processors as we assumed in the Cholesky decomposition time estimate. But beyond one thousand processor, this might not scale, and the time given by Eq.(10) will probably not be reliable. Thus, we cannot solve this problem by simply increasing P in Eq.(10).

Using the 1000 atom Si example shown in Fig.3, we have tested a different algorithm which can potentially solve this matrix diagonalization problem. One important point is that, during the self-consistent iteration, the matrix H(i,j) are always very close to be a diagonal matrix. Thus, what we have is a small perturbation. One might imagine to use perturbation theory to diagonalize this matrix. However, perturbation theory is not always stable. Instead, we have chosen a conjugate gradient (CG) method. Each processor group will work on one eigen vector $V_i(j)$ which is relevant to its wave function ψ_i (the same i). We have asked the conjugated gradient to minimize

$$E_i = V_i^H (H - h_i)^2 V_i \tag{12}$$

where h_i=H(i,i). Usually, the convergence for such CG steps is slow due to the squaring of the matrix. However, we can use the diagonal part of H: H(j,j) as a preconditioner. Since the matrix H is almost diagonal, this preconditioner is extremely effective. The CG convergences for some of the eigen vectors i and for a few self-consistent iteration steps are shown in Fig.6 for the 1000 Si atom system shown in Fig.3. We see that the convergences almost fall into different categories. For higher self-consistent iteration steps (thus, H is more close to diagonal), the convergence is faster. Besides, lower vector index i also has faster convergence. Overall, we can see that, usually 100-200 CG steps will be enough to yield a converged solution. However, we cautious that more research is needed to study the stability of this method, especially to ensure all vectors can be converged to high accuracy and two neighboring i will not converge to the same state. We have found some slow convergence cases, and also some eigen vectors shifted away from their original values (these two facts are often connected). The eigen vector shifting can be checked by seeing whether $V_i(i)$ element has a dominant weight for the whole vector. If not (it is shifted), then it might be necessary to check the neighboring vector i' which is calculated by the neighboring processor groups. It might be possible to develop iterative schemes which requires the final eigen states to have a high component in $V_i(i)$, so it is not shifted. Note that, this algorithm is very close to the residual minimization algorithm used in solving the Schrodinger's equations of the wave functions. Also note that, in our iterative scheme, it is not necessary to diagonalize the matrix H exactly. As we see, at higher self-consistent iteration steps, this H tends to be diagonal by itself. Thus, although further studies might be necessary, here we have demonstrated that such iterative algorithm is quite feasible.

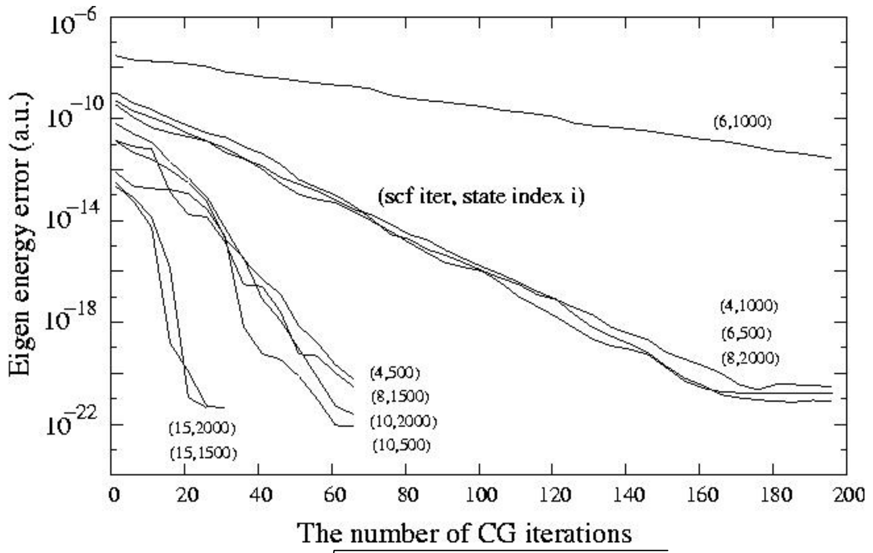

Fig.6, the eigen energy error, defined as $\sqrt{\langle V \mid [H - \langle V \mid H \mid V \rangle]^2 \mid V \rangle}$ as a function of the CG iteration for eigen state V_i of subspace matrix H(scf-iter). The system tested is the 1000 Si atom system shown in Fig.3. Each self-consistent (SCF) iteration in Fig.3 will generate one H(scf-iter). The two numbers in bracket are the indexs for self-consistent iteration and the eigen vector, respectively. For a MD simulation, the SCF iterations are probably correspond to the 7 to 14 SCF iteration shown here (which is started from random wavefunction).

Note that, different processor group will work on V_i for different i, thus there is no repetition for the whole machine. The computational cost is thus 400 matrix-vector multiplications [each $(H-h)^2V$ is obtained by multiplying (H-h) twice]. This computing cost will be distributed among the Np processors within a group. As a result, the floating point operation requirement for each processor is: $400*M^2/Np/0.001s = 1.6$ Gflops. This is a rather small. By distributing the matrix-vector multiplication $W=(H-h)^2V$, each processor will only have M/Np=2 elements (two complex numbers) of the resulting W vector. There is a special task of distributing these two elements (32 Byte) to all the other processors in the group. The time allowed to do this is: 0.001 s /200=5 μs. The success to accomplish this task might critically depend on how it is implemented. In a very crude way, this task can be converted to a global sum of the whole vector W (with each processor has two elements in its initial value). We can follow the in-group global sum argument in Sec. IV(2) for P(R,L) in the nonlocal potential calculation. Now, the data amount (the full W vector) is 32KB, but the allowed time is 5 μs. Then the in-group interconnect link bandwidth needs to be: 128 GB/s. This is quite large, a bit impractical. However, this algorithm is quite wasteful since a lot of zeros are communicated at the initial communication steps (we have assumed 20 communication steps for the global sum). Note that, theoretically each processor needs only to receive 32KB within 5 μs, which is 6 GB/s, which is much smaller than 128GB/s. Besides, if we assume 0.1 μs latency time for processor-to-processor communication, 50 such communications are possible within 5 μs. Combining all these factors, we believe this task is achievable if the in-group interconnect bandwidth is 37GB/s, as assumed in the Cholesky decomposition for S matrix using the Scalapack. However, special algorithms (perhaps implemented in low hardware level) need to be designed carefully for this task. Overall, we believe, with the given architecture requirement discussed above, this CG method will enable us to diagonalize the matrix.

(6) Other calculations

Potential calculations
The evaluation of Eq.(5) needs only two FFTs. Thus the computational requirement is not large. Besides, one only needs to do this once for each self-consistent iteration, thus the cost is small compared to $H\psi_i$. If ultrasoft pseudopotential is used, then the real space grid for potential can be twice as big (in each direction) as the grid used for the wave functions. But even this should not cause any big problem. Besides, the whole machine can be used for this evaluation, not just one group especially if generalized gradient approximation (GGA) is used to calculate the exchange correlation potential, where many FFTs are needed. In that case, different processor groups can calculate different FFTs.
We also don't expect the charge mixing (or potential mixing) step takes time because the total amount of data involved is small.

Atomic force calculations
After the self-consistent solutions are reached, the force on each atom can be calculated using the Hellman-Feynman theory. This can be done relatively easily. The most time assuming part comes from the wave function projector for the nonlocal potential. The total computational cost is similar to that of Eq.(7). If each processor group calculates the projections for all the atoms with the wave function ψ_i in this group, then there need to be a global summation from all the processor groups. This can be done by passing the forces of the 1000 atoms through the Ng groups in a round-robin fashion. We can reduce the data among significantly by passing only the force of one atom from one processor (in a horizontal line of Fig.5) within a processor group. Besides, the atomic forces need to be evaluated only once every MD step, thus overall, the requirement from this step is minimum.

V. CONCLUSION

We have listed the architecture requirements in Table.III derived from different tasks listed in Table.I. In order to achieve the high speed on the proposed 2 million processor computer, we have to adjust our algorithms accordingly. Overall, the P-orth-ψ is the most expensive task both in terms of floating point operation and data communication. If we use the residual minimization algorithm for solving the Schrodinger's equation through out the MD steps, the algorithm can be twice as fast as the all-band conjugate gradient method due to the elimination of this P-orth-ψ task. However, the all-band conjugate gradient method is more reliable. If the all-band conjugate gradient method is used, this P-orth-ψ task needs to take half of the whole computational time in order to have balanced hardware requirements from different parts of the algorithm. For nonlocal potential implementation, we need to treat each atom (and all its angular momentum projection functions) within one processor, instead of using the distributed treatment. Otherwise, the nonlocal potential task will also pose a higher hardware requirement than the rest parts of the program. Finally, in the subspace diagonalization, the MxM matrix H needs to be diagonalized with an iterative method using conjugate gradient method. The current Scalapack will take too long to diagonalize this matrix. On the other hand, to perform Cholesky decomposition for the overlap matrix S, the Scalapack routine is fast enough.

From Table.III, we see that the FFT does not present any hardware challenge. This confirms our initial believe that plane wave method is a good choice. Since some of the most stringent requirements come from wave function communication between processor groups, if real space grid method is used for the calculation, for the same accuracy, 5 times larger amount of wave function data needs to be communicated. Then the inter-group connection bandwidth requirement will be 5 times larger. For all the tasks (except for P-orth-ψ if we request it to take 1/10 of the whole computing time), the demand for CPU speed is not high. This is understandable. As we discussed in the introduction, current VASP program can do one MD step within one minute for a 1000 atom problem on 1000 processors. Here, we are working on the same size problem (thus essentially the same amount of floating point operations), requesting 1000 fold speed up. On the other hand, we have 2000 times more processors. Thus, floating point wise, the request for the CPU speed is the same as the current day computer. The challenge for our strong scale computation is the data communication as we have the data fragmentation 2000 times worse than the current day VASP calculation [14]. Thus, our communication bandwidth and latency have requirement more than those used in current day computers. We also require a careful mapping from the data layout in software to the hardware architecture. Another important feature for our proposed computer is to use the cache more effectively. Due to the massively distribution, the data amount on each processor is also small. This can be taken advantage with by storing the wave function (belonging to each processor) in the L1 cache through out the whole calculation. This will increase the floating point operation speed (thus, no difference between blas2 and blas3 routines), and the communication speed.

One interesting feature evidence in Table.III is that, as the computation becomes very fragmented, besides the P-orth-ψ task, some of the most stringent hardware requirements come from the handling of the MxM dimension matrices (e.g, the overlapping S matrix, its decomposition U, and the subspace Hamiltonian matrix H and its vectors V and W). In terms of total amount of data, they are 1000 times less than the wave functions. However, because each processor group will possess the same matrices as other processor groups, there is a 2000 time redundancy. Such redundancy is necessary because it is hard to treat such matrices (with 4 million elements) concurrently by the 2 million processors of the whole machine. Otherwise the communication latency requirement will be even stronger.

Table.III, summary of the hardware requirement in order to carry out each computational task listed in Table.I. The requirement for the $\left\langle P_i \mid \psi_j \right\rangle$ projection is based on the assumption that it will take half of the total computational time. It is only used in the all-band conjugate gradient scheme, not used in the residue minimization scheme. The third column describes what is the main reason for a given hardware requirement.

FFT (inside one processor group)		
CPU speed	2.4 Gflops	1D FFT
In-group bisection bw	240GB/s	For data transpose
In-group all-to-all	48byte within 100 μs	For data transpose
In-group connect latency	0.1 μs	For data transpose
In-group connect inject. rate	0.5GB/s	For data transpose
L1 cache	300 KB	To store real-space wavefunc.
In-group topology	10x10x10 3D torus	For data trans. and global sum
Nonlocal (one atom per processor algorithm)		
CPU speed	3.5 Gflops	Projection multiplication
In-group connect bw	0.75GB/s	Send wavefunc. to diff. proc.
L2 cache	3MB	To store projector $\Phi_{R,L}$
< P | ψ > projection (assuming it take ½ of the whole computing time)		
CPU speed	4 Gflops	For calc. <P|ψ>
Inter-group link bw	32 GB/s	Transfer wavefunct.
Inter-group link latency	0.5 μs	Transfer wavefunct.
Wave function orthogonalization		
CPU speed	5 Gflops	Calc. overlap matrix, wavefunc.
Inter-group link bw	37 GB/s	Transfer wavefunc., round-robin
Inter-group link latency	0.43 μs	Transfer S matrix and wavefunc.
In-group global sum	32KB in 860 μs	For one row of S matrix
In-group connect bw	37 GB/s	Scalapack Cholesky for S
In-group link latency	0.1 μs	Scalapack, can be 3x larger
L1 cache	300 KB	To store S, and U, blocking
Memory size	300 MB	To store wavefunc.
Memory access rate	37 GB/s	If wavefunc is stored, feed CPU
Subspace diagonalization (using CG to diagonalize subspace matrix)		
In-group mutual broadcast	32byte within 5 μs	To distribute W after H*V
In-group connect bandwidth	37 GB/s	For mutual broadcast
In-group connect latency	0.1 μs	For mutual broadcast

Another stringent requirement comes from communication wave functions from the different processor groups. This poses a 37 GB/s bandwidth request for the inter-group connection, and a corresponding 0.43 μs latency request. Due to the amount of data involved, there is no way such request can be reduced. Finally, the memory requirement is 300 MB. This is to provide the flexibility to store all wave functions $\{\psi_j\}$ in a processor. It also provides the capability to store the whole matrices S, U, H in each processor if necessary. Although the use of careful algorithms can avoid this (e.g., only store the corresponding block of S, U in each processor for Scalapack Cholesky decomposition, and only one row or column of H in each processor), but providing this capability should be useful for program development. This is after all a very modest request. Finally, if all the wave functions $\{\psi_j\}$ are stored in the memory (after one round-robin loop), the memory access rate should be 37GB/s to keep it with the CPU speed in the calculation of S and H matrices, and the calculation of new wave functions in Eq.(9) and Eq.(11). This access speed is the same as the inter-group connection bandwidth, this is because another way to do this is not to store the wave function after each round-robin pass, instead, a new round-robin pass is performed whenever all the wave functions are needed. However, the intuition is that, accessing the data from one CPU's own memory maybe easier than accessing it from its neighbore's cache. If this is true, then it makes sense to store the wave functions in the memory. In any case, providing this capability will greatly increase the programming flexibility. Given all the uncertainties in real program performance, it is a big plus to have such leeway.

Table.IV, a summary of the hardware requirement for the whole machine. The third column describes what is the main reason for each hardware requirement.

CPU speed	5 Glops	Orth, subspace diag.
Inter-group link bw	37 GB/s	Transfer wavefunc., round-robin
Inter-group link latency	0.4 μs	Transfer S, H, and wavefunc.
In-group link bw	37 GB/s	Scalapack for S, and W distribution
In-group link latency	0.1 μs	FFT all-to-all, scalapack for S, W distrib.
In-group bisection bw	240GB/s	FFT all-to-all
Memory access rate	37 GB/s	Calculate S, H, keep up with CPU speed
L1 cache size	300 KB	FFT, to store real space array
L2 cache size	3 MB	To store nonlocal projector
Memory size	300 MB	To store wavefunctions and others
In-group topology	10x10x10 3D torus	For all-to-all and global sum
Core per node	1 - 2	For faster interconnect, not to share link

Finally, combining the requests in Table.III from all the tasks, the overall architecture hardware requirements are summarized in Table.IV. Comparing to current supercomputer architectural parameters, the most challenging requirements are: 37 GB/s inter-group interconnect bandwidth; the 37 GB/s in-group interconnect bandwidth; the 0.1 μs point-to-point in-group communication latency (however, it might be possible to relax this to a neighbore-to-neighbore latency in a 3D torus structure depending on the detail implementations of in-group global sum and Scalapack operations); and the 37 GB/s memory access rate. All these requests are pertinent to data communications and access rate, not to floating point operation speed. In such a largely massive parallel computer, floating point operation usually is not a problem. In contrast, data manipulation and communication is a big challenge. Fortunately, we believe all these requirements are within the reach of next generation interconnect and memory access. Another advantage can be taken with is the dedicated nature of the computer. Since only one code is running, and since there is a rigid one-to-one mapping between the data structure and the computer hardware, the communication protocol can be significantly simplified. As a result, the achievable speed can be more close to the real hardware limit, instead of being delayed by the software layers. Again, one recent example is the dedicated Anton computer. Many of the characteristics (short latency and small message size) of the communication in Anton applications are similar to our current application. It is reported that 60 ns point-to-point communication latency can be achieved in Anton [5]. This is half of the time we requested in our proposed computer. Thus, we believe our proposed computer is within the reach of current day technology.

Acknowledgements

This work was supported by DOE-SC-ASCR office under Contract No. DE-AC02-05CH11231, and National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. This research topic was proposed by Dr. John Shalf. There were extensive stimulating discussions with Dr. J. Shalf during the write-up of this paper. There were also helpful discussions with Dr. Andrew Canning. Finally, the author likes to thank Dr. James Demmel for taking the interest to develop new algorithms to diagonalize the subspace matrix, and for pointing out the relevant literatures. Such on-going algorithm developments will not only benefit the computations proposed in this paper, it will also benefit currently existing codes.

References

[1] P. Hohenberg and W. Kohn, Phys. Rev. 136, B864 (1964).
[2] L.W. Wang, "A survey of codes and algorithms used in NERSC material science allocations", LBNL, technical report, LBNL-61051 (2005).
[3] L.W. Wang, "A survey of codes and algorithms used in NERSC chemical science allocations", LBNL, technical report, LBNL-59066 (2006).
[4] S, Gottlieb, Comp. Sci. & Eng. 8, 15 (2006)
[5] D.E. Shaw, et.al., Commun. of the ACM, 51, 91 (2008).
[6] M. Wehner, L. Oliker, J. Shalf, Int. Journal of High Performance Computing Applications, 22, 149 (2008).
[7] A. E. Terrano, "*The QCD machine*" (Academic Press Ltd, London, 1988).
[8] F. Gygi, et.al., *Proc. 2005 ACM/IEEE conf. on supercomputing* (2005).
[9] C. Dellago, P. Bolhuis, F.S. Csajka, and D. Chandler, J. Chem. Phys. 108, 1964 (1998).
[10] A.F. Voter, J. Chem. Phys. 106, 4665 (1997).
[11] G. Goedecker, Rev. Mod. Phys, 71, 1085 (1999).
[12] L.W. Wang, Z. Zhao, J. Meza, Phys. Rev. B 77, 165113 (2008).
[13] http://cms.mpi.univie.ac.at/vasp/
[14] P. Kent, "Computational challenges in nanoscience: an ab initio perspective", presentation in Peta08 workshop, Hawaii (2008).

[15] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, and J.D. Joannopoulos, Rev. Mod. Phys. 64, 1045 (1992).
[16] A. Stathopoulos, S. Ogut, Y. Saad, J. Chelikowsky, and H. Kim, Comput. In Sci. Eng. 19, July/Aug. (2000).
[17] L.W. Wang, "A brief comparison between grid based real space algorithms and spectrum algorithms for electronic structure calculations", LBNL tech. report, LBNL-63794 (2006).
[18] J. E. Pask, B.M. Klein, C.Y. Fong, and P.A. Sterne, Phys. Rev. B 59, 12352 (1999).
[19] http://hpcrd.lbl.gov/~linwang/PEtot/PEtot.html
[20] W. Kohn and L.J. Sham, Phys. Rev. 136, B864 (1964).
[21] A. Canning, "Scalable parallel 3D FFTs for electronic structure codes", Vecpar08, Toulouse (2008).
[22] D. Raczkowski, A. Canning, L.W. Wang, Phys. Rev. B 64, 121101 (2001).
[23] B. Lee and L.W. Wang, "A new all-band algorithm for density functional theory electronic structure calculations" (unpublished).
[24] P. Pulay, J. Comp. Chem. 3, 556 (1982).
[25] VASP manual: http://cms.mpi.univie.ac.at/vasp/vasp/vasp.html
[26] http://www.top500.org/lists/2008/06
[27] "*Cray XT4 datasheet*", Cray Inc. (2006).
[28] A. Canning, L.W. Wang, A. Williamson, A. Zunger, J. Comp. Phys. 160, 29 (2000).
[29] http://www.netlib.org/scalapack/scalapack_home.html