Performance Evaluation of NWChem Ab-Initio Molecular Dynamics (AIMD) Simulations on the Intel® Xeon Phi™ Processor

Eric J. Bylaska[1(✉)], Mathias Jacquelin[2], Wibe A. de Jong[2], Jeff R. Hammond[3], and Michael Klemm[4]

[1] Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, Richland, WA, USA
`eric.bylaska@pnnl.gov`

[2] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
`{mjacquelin,wadejong}@lbl.gov`

[3] Data Center Group, Intel Corporation, Portland, OR, USA
`jeff.r.hammond@intel.com`

[4] Software and Services Group, Intel Deutschland GmbH, Feldkirchen, Germany
`michael.klemm@intel.com`

Abstract. Ab-initio Molecular Dynamics (AIMD) methods are an important class of algorithms, as they enable scientists to understand the chemistry and dynamics of molecular and condensed phase systems while retaining a first-principles-based description of their interactions. Many-core architectures such as the Intel® Xeon Phi™ processor are an interesting and promising target for these algorithms, as they can provide the computational power that is needed to solve interesting problems in chemistry. In this paper, we describe the efforts of refactoring the existing AIMD plane-wave method of NWChem from an MPI-only implementation to a scalable, hybrid code that employs MPI and OpenMP to exploit the capabilities of current and future many-core architectures. We describe the optimizations required to get close to optimal performance for the multiplication of the tall-and-skinny matrices that form the core of the computational algorithm. We present strong scaling results on the complete AIMD simulation for a test case that simulates 256 water molecules and that strong-scales well on a cluster of 1024 nodes of Intel Xeon Phi processors. We compare the performance obtained with a cluster of dual-socket Intel® Xeon® E5–2698v3 processors.

Keywords: Xeon Phi · Many-core · Chemistry · AIMD · Ab-initio · Molecular dynamics

1 Introduction

One of the more computationally demanding scientific simulations used extensively on today's large-scale parallel computers is Ab-initio Molecular Dynamics

(AIMD) [4,5,7,9,10,13,14,20,25,28]. In this type of simulation the motions of the atoms are simulated using Newton's laws in which the forces on the atoms are calculated directly from the electronic Schrödinger equation, or more specifically in this work, the Kohn-Sham Density Functional Theory (DFT) equations [18,24]. These simulations are computationally expensive because the DFT equations, which are already expensive in their own right for systems beyond a few atoms, are solved at every time integration step in the simulation.

For an AIMD simulation to be viable for the scientist, each step in the full DFT calculation must take the order of a second or less [5] to complete. The need for such fast DFT calculations is driven primarily by the fact that the time step of a conventional AIMD simulation can be quite small (~ 0.2 femtoseconds $= 2 \times 10^{-16}$ s) along with the fact that the length of the simulation will need be at least 10 picoseconds. For many chemical processes of interest, the simulations will need to run on the order of nanoseconds (10^{-9} s and larger). A scientific simulation of about 10 picoseconds requires solving 500,000 DFT calculations in sequence, which takes about 5.8 days assuming that a single DFT calculation (time-step) completes within one second, about 50.8 days with a 10-second time step, and about 1/2 year with a 30 second time step. Compared to merely optimizing a molecule or crystal, which require at most a few 100 evaluations, this is extremely expensive.

It should be noted that, for carrying out geometry optimizations only, the need for extremely fast DFT calculations is not as important as calculating larger numbers of atoms. As a consequence the focus of HPC DFT algorithm development has almost exclusively been on weak parallel scaling algorithms that maintain parallel efficiency as the system size grows. In contrast, the focus of HPC AIMD algorithm development has focused on truly strong parallel scaling algorithms, rather than weak parallel scaling, since the time per step needs to be as small as possible.

With the advent of new HPC systems with multiple levels of parallelism composed of many-core CPUs, e. g., the second generation Intel® Xeon Phi™ processor (code-named "Knights Landing", KNL) [29], and connected by high-speed networking, such as the Cray* Aries* network, algorithms can now be developed that take advantage of fast data movement and fast synchronization between threads on the CPUs. These new systems have the potential for improved strong parallel scaling, however, new algorithms need to be developed that can make use of these massively parallel processor architectures [15].

Although the MPI (or MPI-only) model can be used on many of today's architectures with large numbers of cores [16], and in principle can take advantage of the fact that memory is shared, this programming model has several drawbacks. Performance hits can happen using this programming model because of its lack of ability to control memory at the node resulting in a lack of memory coherency, higher latencies, and slower synchronizations. A more suitable approach for developing strong scaling algorithms on large core architectures is to use a hybrid execution model [27], where data movement between nodes is handled by MPI and the data movement and execution within a node is handled by a multi-threading model such as OpenMP* [11,12,23]. The advantages of this model are that synchronizing between threads is faster, extra data movements can be avoided, and the memory footprint is potentially smaller since particular data structures may not need to be duplicated among threads.

In this paper, recent developments of adding thread-level parallelism to the plane-wave density functional theory (DFT) methodology in NWChem are presented [2,4,6,10]. In our current development, thread-level parallelism is integrated into a MPI-only code using OpenMP constructs and threaded mathematical libraries, such as the Intel® Math Kernel Library. Similar efforts are underway with other codes [1], however, to our knowledge our development is at present unique in that the focus is on having ab-initio molecular dynamics simulations AIMD with very fast iteration times (i. e., very small times per AIMD step). The target platform for our work is the NERSC-8 supercomputer "Cori", which employs a mix of Intel® Xeon® and Intel Xeon Phi processors that are connected through the Cray Aries fabric.

2 Prior Work

There are three key kernels in AIMD that need be efficiently parallelized: 3D FFTs, non-local pseudopotential, and Lagrange multiplier kernels that are used for maintaining orthogonality of Kohn-Sham orbitals [4,7,14,22,32].

In the MPI-only parallel AIMD code in NWChem, the parallel efficiency of the 3D FFT is by far the worst performing kernel, and the best algorithms are only able to use N MPI tasks for a 3D FFT of an $N \times N \times N$ grid. The lack of parallel performance of 3D FFTs is well-known [3,8] and is related to the presence of global all-to-all operations. To overcome this bottleneck, algorithms have been developed that distribute the Kohn-Sham orbitals in addition to partitioning the simulated space [5,7,14,32]. This results in a 2D processor geometry of $Np = Np_i \cdot Np_j$ processors or MPI ranks (see Fig. 1).

The drawback of this strategy is that the Lagrange multiplier kernel becomes less efficient as Np_j becomes larger. In general, increasing Np_j significantly improves the efficiency of the 3D FFT and the non-local pseudopotential kernels, while increasing Np_i favors the Lagrange multiplier kernel. Hence, the best parallel performance is found by balancing the individual performance of the three kernels with respect to Np_i and Np_j. It should be noted that clusters of Xeon Phi processors have the potential to improve strong parallel scaling of the 3D FFT, and as a consequence the overall scaling of AIMD, due to improved memory performance of the high-bandwidth and on-package memory. Using multi-threading instead of MPI primitives, synchronization times of the large numbers of threads within the node can be reduced to increase execution efficiency of the kernels.

Recently, we have reported results from adding thread-level parallelism to the AIMD code in NWChem [15]. The work focused on single-node performance and showed promising results for the multi-threaded implementation of the key kernels in the AIMD calculation. It was shown that through careful optimizations of tall-and-skinny matrix products, which are at the heart of the Lagrange

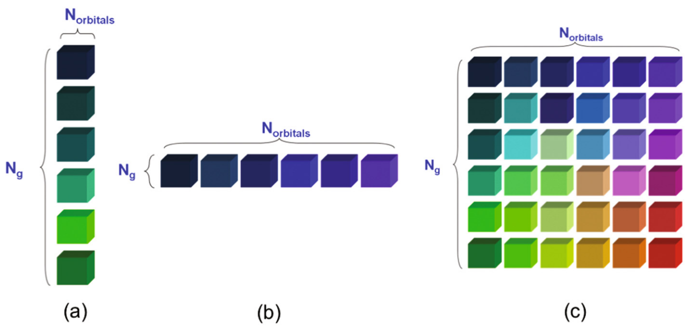

Fig. 1. Possible types of parallel distributions of the Kohn-Sham orbitals in plane-wave DFT software. (a) Each of the Kohn-Sham orbitals is identically spatially decomposed. (b) Each Kohn-Sham orbital is located on different MPI tasks. (c) The 2D-parallel distribution suggested by Gygi et al. [14], where the total Kohn-Sham orbital set are block decomposed.

multiplier and non-local pseudopotential kernels, as well as other optimizations for 3D FFTs, our OpenMP implementation delivered excellent strong scaling for the 68 cores of the Xeon Phi Knights Landing processor. Moreover, it was shown that the straightforward and naive approach of calling a multi-threaded BLAS library from a serial (MPI) rank does not yield a satisfactory level of performance for the Lagrange multiplier and non-local pseudopotential kernels. A roofline model analysis [33] of the Lagrange multiplier verified that our implementation was close to the roofline model of the execution platform for various problem sizes.

3 AIMD Implementation for the Intel Xeon Phi Processor

The bulk of the computational work in AIMD revolves around the solution of N_e eigenvalue equations, $H\psi_i = \epsilon_i\psi_i$, for the electron orbitals ψ_i, appearing as a result of the DFT approximation to the Schrödinger equation.

These eigenvalue equations are subject to orthogonality constraints

$$\int_{\Omega} \psi_i(\boldsymbol{r})\psi_j(\boldsymbol{r})d\boldsymbol{r} = \delta_{i,j} \tag{1}$$

Most standard AIMD algorithms use non-local pseudopotentials and plane-wave basis sets to perform the DFT calculations and are typically solved using a conjugate gradient algorithm or a Car-Parrinello algorithm [4,20]. For DFT, the Hamiltonian operator H may be written as

$$H\psi_i = \left(\begin{matrix} -\frac{1}{2}\nabla^2 + V_l + V_{NL} + V_H[\rho] \\ +V_x c[\rho] \end{matrix} \right) \psi_i \tag{2}$$

$$(-1/2)\nabla^2\Psi + V_{ext}\Psi + V_H\Psi + V_{xc}\Psi = E\Psi$$

$$\langle \Psi_i | \Psi_j \rangle = \delta_{ij}$$

$(-1/2)\nabla^2\Psi :: N_e N_{pack}$

$V_{ext}\Psi :: (N_a N_{pack} + N_g \mathrm{Log} N_{pack} + N_e N_{pack}) + N_a N_e N_{pack}$

$V_H\Psi :: N_e N_g \mathrm{Log} N_g + N_e N_g + 2N_g \mathrm{Log} N_g + N_g + N_e N_g$

$V_{xc}\Psi :: N_e N_g \mathrm{Log} N_g + N_e N_g$

$\langle \Psi_i | \Psi_j \rangle :: N_e^2 N_{pack} + N_e^3$

N_a - number of atoms, N_e - number of electrons
N_g – size of FFT grid, N_{pack}- size of reciprocal space

Fig. 2. Operation count of $H\psi_i$ in a plane-wave DFT simulation.

where the one-electron density is given by

$$\rho(\boldsymbol{r}) = \sum |\psi_i(\boldsymbol{r})|^2 \tag{3}$$

The local and non-local pseudopotentials, V_l and V_{NL}, represent the electron-ion interaction. The Hartree potential V_H is given by

$$\nabla^2 V_H = -4\pi\rho \tag{4}$$

and the exchange and correlation potential is $V_x c$. The algorithmic cost to evaluate $H\psi$ and maintain orthogonality are shown in Fig. 2.

Due to their computational complexity, the electron gradient $H\psi_i$ and orthogonalization need to be calculated as efficiently as possible. The main parameters that determine the cost of a calculation are N_g, N_e, N_a, and N_{proj}, where N_g is the size of the three-dimensional FFT grid, N_e is the number of occupied orbitals, N_a is the number of atoms, N_{proj} is the number of projectors per atom, and N_{pack} is the size of the reciprocal space. Detailed estimates for the scalability of these calculations in terms of the AIMD parameters can be derived and fit in terms of a finite set of rates and bandwidths that are machine dependent (e.g., see Bylaska et al. [5]). Fitting the machine dependent parameters was not performed in this initial parallel benchmark study, because a large number of calculations is needed for accurate fitting.

As shown in Fig. 2, the evaluation of the electron gradient (and orthogonality) contains three major computational pieces:

- *applying* V_H and V_{xc}, involving the calculation of $2N_e$ 3D FFTs;
- *the non-local pseudopotential*, V_{NL}, dominated by the cost of the matrix multiplications $W = P^TY$, and $Y_2 = PW$, where P is an $N_{pack} \times (N_{proj} \cdot N_a)$ matrix, Y and Y_2 are $N_{pack} \times N_e$ matrices, and W is an $(N_{proj}N_a) \times N_e$ matrix;
- *enforcing orthogonality*, where the most expensive matrix multiplications are $S = Y^TY$ and $Y_2 = YS$, where Y and Y_2 are $N_{pack} \times N_e$ matrices, and S is an $N_e \times N_e$ matrix.

In the next subsections, we focus on the main components of an AIMD step that need to be parallelized both at the shared-memory and the distributed-memory levels to achieve good parallel performance. All invocations of MPI primitives in NWChem are done from within an OpenMP `master` region, requiring the `MPI_THREAD_FUNNELLED` threading level to be used [21]. This keeps messages size larger and all threads then work on the same data block that was send once, instead of having each thread communicate a smaller block.

3.1 3D FFTs

For each iteration of an AIMD simulation, N_e Kohn-Sham orbitals, $\psi(G, 1 : N_e)$, are converted from reciprocal space to real space and N_e orbital gradients are transformed from real space to reciprocal space. This corresponds to computing N_e reverse 3D FFTs and N_e forward 3D FFTs. In reciprocal space, only a sphere of radius E_{cut} (or hemisphere for a Γ-point code), and contained within the 3D FFT block, is needed and saved in the program.

Each 3D FFT consists of six distinct steps, each of which is executed for each of the N_e Kohn-Sham orbitals in a pipelined fashion as illustrated in Fig. 3. For the forward 3D FFT, the steps are (in reverse order for backward FFTs):

1. Unpack the reciprocal space sphere into a 3D cube, where the leading dimension of the cube is in the z-direction, second dimension is the x-direction, and the third dimension is the y-direction.
2. Perform $nx \times ny$ FFTs along the z-direction. Note that only the arrays that intersect the sphere need to be computed.
3. Rotate the cube so that the first dimension is the y-direction, $z, x, y \rightarrow y, z, x$.
4. Perform $nz \times nx$ 1D FFTs along the y-direction.
5. Rotate the cube so that the first dimension is the x-direction, $y, z, x \rightarrow x, y, z$.
6. Perform $ny \times nz$ 1D FFTs along the x-direction.

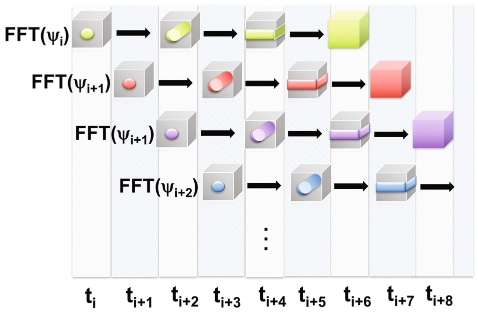

Fig. 3. Illustration of the pipelined 3D FFT algorithm used in the NWChem AIMD code.

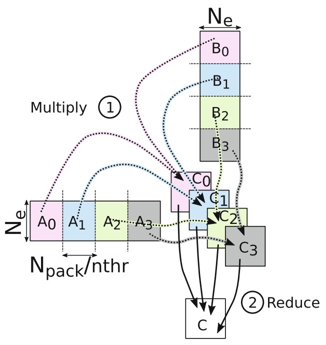

Fig. 4. Multithreading scheme of the ***FFM*** operation.

The 3D FFTs used in this paper were implemented by modifying the existing parallel 3D FFTs contained in the NWChem plane-wave module (called NWPW). More details on the implementation of these FFTs can be found in prior work by Bylaska et al. [5,6,16].

In the initial parallel FFT code, the 3D cube is distributed along the 2nd and 3rd dimension. This distribution is block-mapped using a two-dimensional Hilbert curve spanning the grid of the second and third dimensions (see [6]). This two-dimensional Hilbert parallel FFT was built using a 1D FFT and a parallel block rotation. The FFTPACK library [30] is used to perform the 1D FFTs, and the parallel block rotation was implemented using non-blocking MPI primitives.

We generalized the FFTs to a hybrid MPI-OpenMP model by making the following changes. The planes of 1D FFTs in steps 2, 4, and 6 execute on multiple threads through an OpenMP `DO` directive so that a single 1D FFT is carried out on one thread. The data rearrangement in steps 1, 3, and 5 is threaded using a `DO` directive on the loops that perform the data-copying on the node.

3.2 Lagrange Multipliers and Non-local Pseudopotentials on 1D and 2D Processor Grids

At each step of an AIMD simulation, wave functions need to be orthogonalized. This is the purpose of the Lagrange multiplier method. Details on the algorithm itself can be found in [4,20,26]. The Lagrange multiplier method solves several matrix Riccatti equations [19] at every step. We have introduced a highly scalable multi-threaded implementation of the Lagrange multiplier for the Xeon Phi processor in [15]. In the following, we go through the different steps that were required to derived a scalable hybrid MPI-OpenMP implementation for a distributed memory many-core cluster.

The Lagrange multiplier algorithm can be described as a sequence of matrix-matrix products of different sizes. In [15], we have introduced the following formalism. The letter **F** refers to an $N_{pack} \times N_e$ or an $N_e \times N_{pack}$ matrix, and **M**

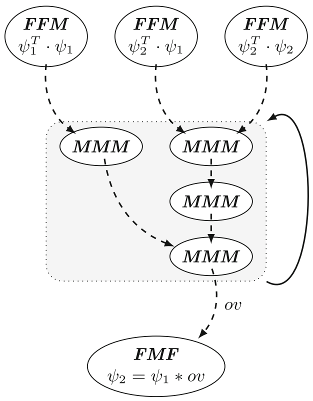

Fig. 5. Dependencies between operations of the Lagrange multiplier algorithm.

refers to an $N_e \times N_e$ matrix. A matrix product $C = AB$ can then be described by a sequence of three letters, the first referring to matrix A, the second to matrix B, and the last one to matrix C. In general, $N_{pack} >> N_e$, thus, $\mathbf{F}$ matrices or their transpose are *tall-and-skinny* matrices.

The Lagrange multipliers method requires three types of matrix product to be computed: ***MMM***, ***FMF***, and ***FFM***. The dependencies between these operations is depicted in Fig. 5.

The ***FFM*** type of matrix product is the most expensive part of the Lagrange multiplier method. For this particular matrix shape, multi-threaded implementations available in vendor libraries do not scale well. In [15], we have introduced an OpenMP algorithm that scales better than vendor solutions and that blends well with the outer-level parallelism of an active OpenMP `parallel` region. The parallelization scheme used in this approach is depicted in Fig. 4.

In order to exploit distributed memory processor grids (cf. Sect. 2), we have implemented a version of the Scalable Universal Matrix Multiplication Algorithm (SUMMA) [31]. The rationale behind SUMMA is to leverage the efficiency of MPI collective communications. When computing $C = AB$ on a $Np_i \times Np_j$ 2D process grid, SUMMA broadcasts the current block of matrix A within row of processes and the current block of B across a column a processes. The product between these two blocks is then added to the local block of C. These local contributions are computed using the OpenMP multi-threaded algorithm introduced in [15]. In the case of the ***FFM*** operation, matrix A is of size $N_e \times N_{pack}$, and is distributed over a $Np_j \times Np_i$ grid of MPI ranks. The C matrix is $N_e \times N_e$ and is replicated over $Np = Np_i \cdot Np_j$ copies. SUMMA is applied followed by a global reduction of C to produce replicated matrices.

As part of the non-local pseudopotentials computation, a sequence of ***FFM***, ***MMM***, and ***FMF*** matrix products also need to be computed. This is similar to the Lagrange multipliers method except that $\mathbf{M}$ refers to a $(N_a N_{proj}) \times N_e$ matrix and $\mathbf{F}$ refers to either a $N_{pack} \times N_e$ or $N_{pack} \times (N_a N_{proj})$ matrix, where N_a is the number of atoms and N_{proj} is the average number of projectors per atoms. For most systems, N_e is approximately $N_a N_{proj}$. Note that the matrix operations for non-local pseudopotentials (and Projector Augmented Wave projectors) are separable across atoms (and, for some pseudopotentials, separable across projectors), that is, C is block diagonal between atoms, and can be evaluated atom by atom, although blocking is usually done to improve the efficiency.

4 Performance Evaluation

Our evaluation of the plane-wave DFT AIMD method has been performed on the "Cori" system at NERSC. We use both partitions of the system to compare the performance of the new code on both Intel Xeon processors (codename "Haswell") and Intel Xeon Phi processors (codename "Knights Landing" or KNL for short).

The "Haswell" partition is a Cray XC40 system and consists of 2388 dual-socket nodes with Intel Xeon E5-2698v3 processors running 16 cores per socket.

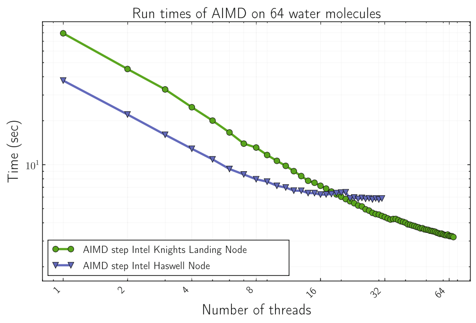

Fig. 6. Scalability of the multi-threaded NWPW code within a single Xeon and Xeon Phi node for 64 water molecules.

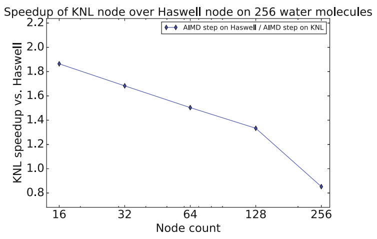

Fig. 7. Speedups of the Xeon Phi processors over the Xeon processors the "water256" benchmark at different numbers of nodes.

The nodes are configured without Hyper-Threading, run at frequency of 2.3 GHz, and are equipped with 128 GB of DDR4 memory with 2133 MHz. The nodes are connected through the Cray Aries interconnect with Dragonfly topology [17].

The "Knights Landing" partition is also a Cray XC40 system with 9688 single-socket nodes with Intel Xeon Phi 7250 processors. Each processor features 68 cores with four hardware threads per core. The cores are running at a frequency of 1.4 GHz. Each node contains 96 GB of DDR4 memory running at 2400 MHz. For our evaluation, the 16 GB on-package high-bandwidth memory has been configured to run in quadrant mode and is used in cache mode [29]. Similarly to the Xeon partition, the Xeon Phi partition uses the Cray Aries interconnect with the Dragonfly topology.

In order to compare the performance of a node of the Xeon partition of Cori to that of a node of the Xeon Phi partition, we conducted an intra-node strong scaling study with a benchmark that simulates 64 water molecules ("water64"). The pertinent dimensions for this system are $N_e = 512$, $N_g = 1,259,712$ (108^3) and $N_{pack} = 106,456$. To assess cluster performance, we use a larger input deck that simulates 256 water molecules ("water256"). The matrix dimensions for this system are $N_e = 2056$, $N_g = 5,832,000$ (180^3) and $N_{pack} = 437,600$. We run the "water256" benchmark on up to 256 nodes of the Xeon partition to establish the baseline performance and to compare it with the Xeon Phi nodes. We then scale the benchmark to up to 1024 nodes of the Xeon Phi partition to show the feasibility of the AIMD plane-wave algorithm at a large-scale many-core system.

The first set of experiments aims at comparing the single-node performance of the processors. This gives insight into the relative speed-up of the Xeon Phi processor over the Xeon processor without the effects of the interconnect fabric. Increasing number of threads from one to the maximum available physical cores, we observe that a Xeon Phi node achieves a 1.8x speedup over a Xeon node when used at full capacity (see Fig. 6). The Haswell processor shows a flat performance profile at about 16 threads, as it reaches its memory-bandwidth limits, whereas

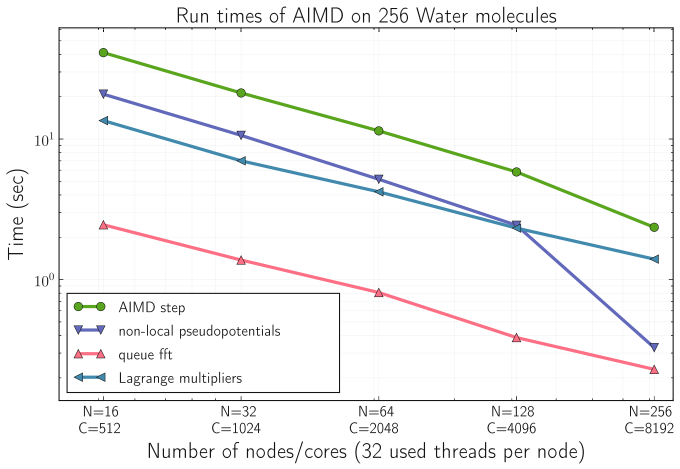

Fig. 8. Scalability of major components of an AIMD step on the Xeon partition for "water256".

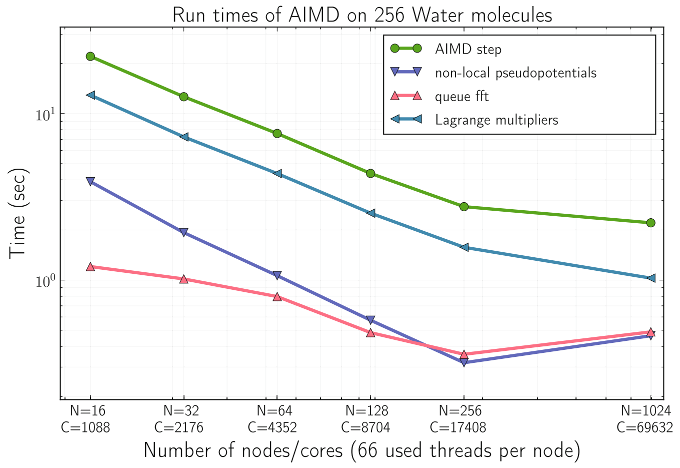

Fig. 9. Scalability of major components of an AIMD step on the Xeon Phi partition for "water256".

the Knights Landing processor still provides a speed-up with all physical cores utilized due to the high-bandwidth on-package memory.

Next, we compare results obtained of the "water256" benchmark on 16, 32, 64, 128, and 256 nodes of each partition. We use 32 threads per Xeon node and 66 threads per Xeon Phi node. Leaving two cores of the Xeon Phi processor for the operating system is best for performance. The Xeon Phi partition achieves a speedup of 1.86x, 1.68x, 1.5x, and 1.3x over the Xeon partition on 16, 32, 64, and 128 nodes (see Fig. 7), showing that NWChem is able to exploit the additional

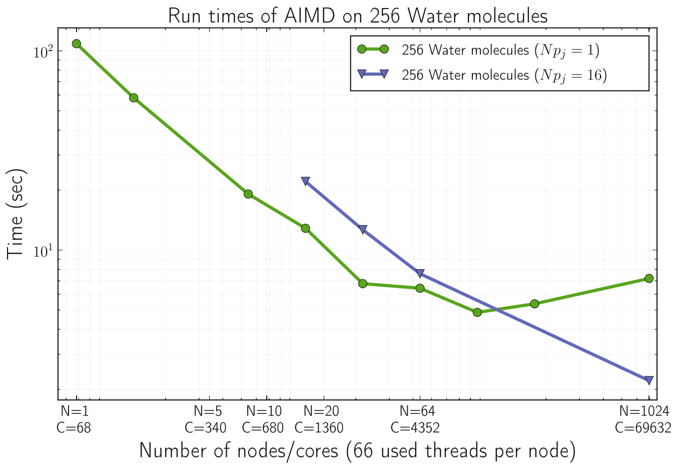

Fig. 10. Scalability of AIMD on 256 water molecules ($Np_j = 1$ and $Np_j = 16$).

computing power of Xeon Phi nodes. It is important to note that as the number of nodes grows, the local amount of work per node is reduced while the impact of the interconnect and communication increases. This explains why the speedup of KNL nodes over Xeon nodes decreases when the number of nodes increases. Ultimately, the local work per node becomes too small to occupy the network fully and thus the performance advantage of the Xeon Phi processor vanishes. When using 256 nodes, Xeon nodes become faster than Xeon Phi nodes in this latency-limited regime due to the 2x higher clock frequency, 0.5x flops/cycle, and 0.25x bytes/sec of the Xeon nodes. Figure 8 and Fig. 9 illustrate the scalability of the most expensive components of the AIMD simulation, which are the calculation of the FFTs, the Lagrange multipliers method, and the non-local pseudopotentials computation. The results show that all components scale well on both Xeon and Xeon Phi nodes. However, it is interesting to note that due to the differences in the underlying processor architectures, the relative cost of each component is different on Xeon and Xeon Phi. The computation of non-local pseudopotentials and Lagrange both dominate the cost on Xeon, while the Lagrange multiplier is the dominating kernel on Xeon Phi. Computing the non-local pseudopotentials is a similar process as the Lagrange multiplier, with fewer intermediate steps between the ***FFM*** operations. As the Xeon Phi nodes use more threads than the Xeon nodes, the N_{pack} dimension is split in smaller blocks. Therefore, each thread receives a smaller amount of work. For the Lagrange multipliers method, the dependencies between the ***FFM*** operations and the ***MMM*** operations are such that the effect of this trend is less visible (see Fig. 5).

Figure 10 shows the effect of changing the processor grid by increasing the Np_j dimension, as briefly described in Sect. 2. As can be seen from Fig. 10, the scalability of the computation can be improved from 128 to 1024 nodes by balancing Np_i and Np_j to favor the calculation of 3D FFTs and non-local pseudopotentials. For the sake of brevity, we did not fully explore the full parameter space of node counts and the shape of the $(Np_i \times Np_j)$-grid distribution. We plan to conduct such an analysis as a future work.

5 Conclusion and Future Work

The parallelism available on machines with many-core processors requires to revisit the implementation of their programs to efficiently use the available resources. In this paper, we have demonstrated that rewriting key kernels in NWChem's plane-wave AIMD module to use a hybrid MPI-OpenMP that provides good scalability on a many-core cluster based on the Intel Xeon Phi Processor. However, to achieve this level of performance for large AIMD simulations the parallelism within a node must be implemented at a very fine grain level and needs careful orchestration of MPI-level parallelism and OpenMP threading.

The unique implementations of key kernels used in AIMD such as sphere to cube 3D FFTs and the matrix multiplication of tall-skinny matrices require special attention and are not well for suited standard computational math libraries. For example, due to the shape the matrices, standard BLAS libraries such as Intel Math Kernel Library have a hard time to provide close-to-optimal performance on a many-core system. However, by rewriting these kernels from scratch using the hybrid MPI-OpenMP model at a required very fine grain level we were able to obtain good performance.

For this paper, we simulated up to 256 water molecules, a standard benchmark for AIMD, to test our implementation. The experiments showed strong scaling up to 1024 KNL nodes (69632 cores) for 256 water molecules. The timings of the major kernels, the pipelined 3D FFTs, non-local pseudopotential, and Lagrange multiplier kernels all displayed significant speedups. Further, comparisons between the KNL and Haswell nodes showed that that Xeon Phi partition was able to attain more than 1.5x speedup over the Xeon partition.

As future work, we plan to implement hybrid MPI-OpenMP algorithms for exact exchange kernels needed for hybrid DFT calculations, as well as propagate our current developments into the band structure code in NWChem. We also plan to explore the parameter space in more detail and determine the best setting of Np_i and Np_j for various node counts. Lastly, we are planning to carry out runs at scale of a multiple of thousands of many-core cluster nodes to simulate a problem that is of interest to the chemistry and geochemistry communities.

Acknowledgment. This work was supported by the NWChem project in the William R. Wiley Environmental Molecular Sciences Laboratory (EMSL), the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research ECP program (NWChemEx project), and E.J.B was also supported by the the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Chemical Sciences, Geosciences, and Biosciences Division at PNNL, DE-AC06-76RLO 1830. EMSL operations are supported by the DOE's Office of Biological and Environmental Research. M.J. and W.A.D. were partially supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences. In particular, M.J. was supported by the FASTMath SciDAC institute. We wish to thank the Scientific Computing Staff, Office of Energy Research, and the U. S. Department of Energy for support through the NERSC NESAP program the National Energy Research Scientific Computing Center (Berkeley, CA). This work was also supported by Intel as part of its Intel Parallel Computing Centers effort. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

1. Measuring arithmetic intensity, http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/. Accessed 22 Oct 2016
2. Aprà, E., Bylaska, E.J., Dean, D.J., Fortunelli, A., Gao, F., Krstić, P.S., Wells, J.C., Windus, T.L.: NWChem for materials science. Comput. Mater. Sci. **28**(2), 209–221 (2003)
3. Ayala, O., Wang, L.P.: Parallel implementation and scalability analysis of 3D fast fourier transform using 2D domain decomposition. Parallel Comput. **39**(1), 58–77 (2013). http://www.sciencedirect.com/science/article/pii/S0167819112000932
4. Bylaska, E., Tsemekhman, K., Govind, N., Valiev, M.: Large-scale plane-wave-based density functional theory: formalism, parallelization, and applications. In: Computational Methods for Large Systems: Electronic Structure Approaches for Biotechnology and Nanotechnology, pp. 77–116 (2011)
5. Bylaska, E.J., Glass, K., Baxter, D., Baden, S.B., Weare, J.H.: Hard scaling challenges for ab initio molecular dynamics capabilities in nwchem: using 100,000 CPUs per second. In: Journal of Physics: Conference Series, vol. 180, p. 012028. IOP Publishing (2009)

6. Bylaska, E.J., Valiev, M., Kawai, R., Weare, J.H.: Parallel implementation of the projector augmented plane wave method for charged systems. Comput. Phys. Commun. **143**(1), 11–28 (2002)
7. Canning, A., Raczkowski, D.: Scaling first-principles plane-wave codes to thousands of processors. Comput. Phys. Commun. **169**(1), 449–453 (2005)
8. Canning, A., Shalf, J., Wang, L.W., Wasserman, H., Gajbe, M.: A comparison of different communication structures for scalable parallel three dimensional FFTs in first principle codes. In: Chapman, B., Desprez, F., Joubert, G.R., et al. (eds.), pp. 107–116 (2010)
9. Car, R., Parrinello, M.: Unified approach for molecular dynamics and density-functional theory. Phys. Rev. Lett. **55**(22), 2471 (1985)
10. Chen, Y., Bylaska, E., Weare, J.: First principles estimation of geochemically important transition metal oxide properties. In: Molecular Modeling of Geochemical Reactions: An Introduction, p. 107 (2016)
11. Cramer, T., Schmidl, D., Klemm, M., an Mey, D.: OpenMP programming on Intel Xeon Phi Coprocessors: an early performance comparison. In: Proceedings of Many Core Applications Research Community (MARC) Symposium, pp. 38–44 (2012)
12. Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. IEEE Computat. Sci. Eng. **5**(1), 46–55 (1998)
13. Fattebert, J.L., Osei-Kuffuor, D., Draeger, E.W., Ogitsu, T., Krauss, W.D.: Modeling dilute solutions using first-principles molecular dynamics: computing more than a million atoms with over a million cores. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, pp. 12–22. IEEE (2016)
14. Gygi, F.: Architecture of Qbox: A scalable first-principles molecular dynamics code. IBM J. Res. Develop. **52**(1.2), 137–144 (2008)
15. Jacquelin, M., De Jong, W., Bylaska, E.: Towards highly scalable Ab initio molecular dynamics (AIMD) simulations on the Intel knights landing manycore processor. In: 31st IEEE International Parallel & Distributed Processing Symposium. IEEE Computer Society (2017, Accepted)
16. de Jong, W.A., Bylaska, E., Govind, N., Janssen, C.L., Kowalski, K., Müller, T., Nielsen, I.M., van Dam, H.J., Veryazov, V., Lindh, R.: Utilizing high performance computing for chemistry: parallel computational chemistry. Phys. Chem. Chem. Phys. **12**(26), 6896–6920 (2010)
17. Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-driven, highly-scalable dragonfly topology. SIGARCH Comput. Archit. News **36**(3), 77–88 (2008). http://doi.acm.org/10.1145/1394608.1382129
18. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. **140**(4A), A1133 (1965)
19. Lancaster, P., Rodman, L.: Algebraic Riccati Equations. Clarendon Press, Oxford (1995)
20. Marx, D., Hutter, J.: Modern methods and algorithms of quantum chemistry. Grotendorst, J. (ed.), pp. 301–449 (2000)
21. MPI Forum: MPI: A Message-passing Interface Standard. Tech. rep., June 2015
22. Nelson, J., Plimpton, S., Sears, M.: Plane-wave electronic-structure calculations on a parallel supercomputer. Phys. Rev. B **47**(4), 1765 (1993)
23. OpenMP Architecture Review Board: OpenMP Application Program Interface, Version 4.5, November 2015. http://www.openmp.org/

24. Parr, R.G.: Density functional theory of atoms and molecules. In: Fukui, K., Pullman, B. (eds.) Horizons of Quantum Chemistry. Académie Internationale Des Sciences Moléculaires Quantiques/International Academy of Quantum Molecular Science, vol. 3, pp. 5–15. Springer, Dordrecht (1980). doi:10.1007/978-94-009-9027-2_2
25. Payne, M.C., Teter, M.P., Allan, D.C., Arias, T., Joannopoulos, J.: Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. Rev. Mod. Phys. **64**(4), 1045 (1992)
26. Polian, A., Loubeyre, P., Boccara, N.: Simple molecular systems at very high density. In: NATO Advanced Science Institutes (ASI) Series B, vol. 186 (1989)
27. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 427–436. IEEE (2009)
28. Remler, D.K., Madden, P.A.: Molecular dynamics without effective potentials via the car-parrinello approach. Mol. Phys. **70**(6), 921–966 (1990)
29. Sodani, A.: Knights landing (KNL): 2nd Generation Intel® Xeon Phi Processor. In: Presentation at Hot Chips: A Symposium on High Performance Chips, August 2015
30. Swarztrauber, P.: Fftpack: a package of fortran subprograms for the fast fourier transform of periodic and other symmetric sequences. Obtainable by e-mail or by ftp from nctlib@ornl.gov (1985)
31. Van De Geijn, R.A., Watts, J.: Summa: scalable universal matrix multiplication algorithm. Concurrency-Pract. Exp. **9**(4), 255–274 (1997)
32. Wiggs, J., Jonsson, H.: A hybrid decomposition parallel implementation of the car-parrinello method. Comput. Phys. Commun. **87**(3), 319–340 (1995)
33. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009). http://doi.acm.org/10.1145/1498765.1498785