THEME ARTICLE: THE EXCALIBUR PROJECT

Portable Acceleration of Materials Modeling Software: CASTEP, GPUs, and OpenACC

Matthew Smith and Arjen Tamerus, *University of Cambridge, CB3 0HE, Cambridge, U.K.*

Phil Hasnip, *University of York, YO10 5DD, York, U.K.*

In this article, we present work to port the CASTEP first-principles materials modeling program to accelerators using open accelerator (OpenACC). We discuss the challenges and opportunities presented by graphical processing units (GPU) architectures in particular, and the approach taken in the CASTEP OpenACC port. Whilst the port is still under active development, early performance results show that significant speed-ups may be gained, particularly for materials simulations using so-called "nonlocal functionals," where speed-ups can exceed a factor of ten.

The introduction first-principles materials modeling is an invaluable tool for scientists to investigate the chemical, physical, and electronic properties of matter, especially in the solid-state.[2] The term "first-principles" refers to any of a number of methods based on parameter-free quantum mechanical models; of these, the most ubiquitous is density functional theory (DFT), especially in the popular plane-wave pseudopotential approach.

At the heart of a DFT simulation is the solution of the Kohn–Sham equations,[1] which for a material take the form of a set of nonlinear eigenvalue problems. Each eigenvalue problem corresponds to a particular choice of the wavevector (called a "$\mathbf{k}$-point") of the particle wavefunctions; for a particular choice of $\mathbf{k}$, we have

$$\mathbf{H}_\mathbf{k}[\rho]\psi_{b\mathbf{k}}(\mathbf{r}) = E_{b\mathbf{k}}\psi_{b\mathbf{k}}(\mathbf{r}) \tag{1}$$

where $\mathbf{H}_\mathrm{k}[\rho]$ is the so-called Hamiltonian matrix, $\psi_{b\mathrm{k}}$ and $E_{b\mathbf{k}}$ are the spatially varying Kohn–Sham wavefunction and energy corresponding to particles in state b (known as "bands") with wavevector $\mathbf{k}$, and ρ is the probability density of the electrons (also spatially varying).

There are solutions to (1) for any particular choice of the wavevector $\mathbf{k}$, and the general solution is found by integrating (numerically) over the possible wavevectors (those within the first Brillouin zone). Thus, the probability density of the electrons is

$$\rho(\mathbf{r}) = \sum_{b\mathrm{k}} f_{b\mathrm{k}} |\psi_{b\mathrm{k}}(\mathbf{r})|^2 \tag{2}$$

where $f_{b\mathrm{k}}$ is the probability of a particle being in band b with wavevector $\mathbf{k}$, and $|\psi_{b\mathrm{k}}(\mathbf{r})|^2$ is the probability density of that band. For fermions, such as electrons, no two particles may be in the same state, so $0 \leq f_{b\mathrm{k}} \leq 1$. The summation over b accounts for the contributions from all the possible states of the particles; the summation over discrete $\mathbf{k}$ is an approximation to the integral over all $\mathbf{k}$.

At first sight, the equations at different $\mathbf{k}$-points in (1) appear independent; however, since the Hamiltonian matrix depends on ρ, and ρ involves a summation over $\mathbf{k}$ [see (2)], the equations are coupled, albeit not tightly. Moreover, the solution of (1) requires knowledge of the density $\rho(\mathbf{r})$ in order to yield $\psi_{b\mathrm{k}}(\mathbf{r})$, yet $\rho(\mathbf{r})$ itself *depends* on $\psi_{b\mathrm{k}}(\mathbf{r})$. For these reasons, the usual approach is to solve (1) and (2) iteratively, starting from an initial guess for $\rho(\mathbf{r})$ and $\psi_{b\mathrm{k}}(\mathbf{r})$.

PLANE-WAVE BASIS SET

The size of the eigenvalue problem depends on the representation of $\psi_{b\mathrm{k}}(\mathbf{r})$. For materials modeling, a common approach is to exploit the periodicity of the electronic unit cell of the material and express all the quantities in a Fourier basis. The density, $\rho(\mathbf{r})$, is cell-periodic (i.e., has the same periodicity as the electronic unit cell), and the complex-valued wavefunctions are "quasi-periodic"; that is, their magnitudes are cell-periodic, as may be seen from (2), but the periodicity of their phase is determined by the wavevector $\mathbf{k}$. It

Digital Object Identifier 10.1109/MCSE.2022.3141714
Date of publication 10 January 2022; date of current version 14 March 2022.

is convenient to express $\psi_{\mathrm{bk}}(\mathbf{r})$ as the product of the phase-factor $e^{i\mathbf{k}\cdot\mathbf{r}}$ and a cell-periodic magnitude, which may be expanded in a Fourier basis. In 3-D, the Fourier basis components are *plane-waves*, and the wavefunctions are expressed as

$$\begin{aligned}\psi_{\mathrm{bk}}(\mathbf{r}) &= e^{i\mathbf{k}\cdot\mathbf{r}}\sum_{\mathbf{G}} c_{\mathbf{G}b\mathbf{k}}e^{i\mathbf{G}\cdot\mathbf{r}}\\ &= \sum_{\mathbf{G}} c_{\mathbf{G}b\mathbf{k}}e^{i(\mathbf{G}+\mathbf{k})\cdot\mathbf{r}} \end{aligned} \tag{3}$$

where $c_{\mathbf{G}b\mathbf{k}}$ is a complex coefficient and each $\mathbf{G}$ is a reciprocal lattice vector (often known as a "G-vector"); reciprocal lattice vectors are wavevectors for which $e^{i\mathbf{G}\cdot\mathbf{r}}$ has the periodicity of the material.

Any linear combination of reciprocal lattice vectors gives the correct periodicity, and the complete basis set is, therefore, infinite. As $|\mathbf{G}|$ increases, however, the corresponding coefficients tend asymptotically to zero, and thus, the expansion may be truncated safely at some cut-off wavevector magnitude, G_c. Denoting the number of plane-waves in the basis set as N_p, each wavefunction is completely determined by the set of N_p complex coefficients, and the Hamiltonian is a complex Hermitian matrix of order $N_p \times N_p$.

In principle, (1) may now be solved via standard matrix diagonalization methods, which would yield all N_p eigenstates and eigenvalues of the Hamiltonian. The plane-wave basis set is computationally efficient for many operations, including differentiation and integration, but N_p is usually large and grows with the size of the simulated volume. Simulations of small systems typically require several thousand plane-waves, and large research simulations may comprise millions of plane-waves; thus, direct diagonalization of the $N_p \times N_p$ Hamiltonian matrix is computationally expensive (direct diagonalization methods scale as the cube of the matrix size, N_p^3). Furthermore, direct diagonalization yields all N_p eigenstates, which is typically two orders of magnitude more than the $\sim N$ required to model the behavior of the N particles. For these reasons, when using a plane-wave basis set, it is common to use an iterative diagonalization method, which allows the calculation to be restricted to the $\sim N$ eigenstates of interest.

In iterative diagonalization, a set of $\sim N$ trial eigenvectors is improved by successive iterations until appropriate convergence criteria are satisfied. There are many possible iterative methods for solving eigenvalue problems; two important classes of approach are subspace methods (e.g., block-Davidson, Arnoldi) and quasi-Newton methods (e.g., conjugate gradients, L-BFGS). Almost all of these methods proceed by repeated application of a matrix to the set of trial states and, in the present context, do not require the construction and storage of the Hamiltonian matrix explicitly, only the ability for it (and related matrices) to be applied to trial eigenvectors.

KOHN–SHAM HAMILTONIAN MATRIX

The Hamiltonian at each $\mathbf{k}$-point comprises the following three core terms: the kinetic energy of the particles (T), the local potential (V_{loc}), and a nonlocal potential (V_{nl})

$$\mathrm{H}_{\mathbf{k}}[\rho] = T + V_{\mathrm{loc}} + V_{\mathrm{nl}}. \tag{4}$$

The local potential describes not only the electron-nuclei Coulomb attraction and interelectron Coulomb repulsion, but also the purely quantum mechanical *exchange–correlation* interaction between the electrons themselves. The nonlocal potential mimics the effect of the innermost ("core") electrons, allowing the computational effort to be focused only on the outermost ("valence") electrons, which are the chemically and electronically active ones. In this way, both the Coulomb attraction of the valence electrons to the nuclei and the Coulomb and exchange–correlation interaction with the core electrons are replaced by an effective interaction with composite "ions"; the resultant effective potential is known as a "pseudopotential."

The kinetic energy matrix is diagonal in Fourier space, which is the native space when working in a plane-wave basis. In contrast, the local potential matrix is diagonal in direct space, so the most efficient algorithm is to transform the wavefunction to direct space (an inverse Fourier transform), apply the local potential, and then Fourier transform back into the plane-wave basis. The nonlocal pseudopotential matrix is represented as a low-rank matrix update, and may be applied efficiently in either space (Fourier or direct); in this work, we apply the matrix update in Fourier space.

The exact form of the electron–electron interaction is not known due to the exchange–correlation component, and so must be approximated in practice. The Hamiltonian expression given in (4) corresponds to an important class of approximations known as *semilocal* exchange–correlation functionals. When using an approximation of this form, the exchange–correlation potential at a point in space depends only on the density, and perhaps its derivative, at that point in space, and it may be included in V_{loc}. An alternative approach generalizes the Kohn–Sham method and introduces a class of *nonlocal exchange–correlation* (NLXC) approximations. The Hamiltonian subsequently acquires an extra potential term (V_{nlxc}), which is additive and fully nonlocal.

CASTEP

CASTEP is a leading implementation of the plane-wave pseudopotential DFT method.[4] It was originally developed in the 1980s and 1990s, but was completely redesigned and rewritten from the ground up from 1999 to 2001 with a focus on usability, portability, and parallel efficiency. The first official release of the new CASTEP was in 2001, and it has been developed continually ever since. CASTEP simulations annually support over 1000 peer-reviewed publications in the scientific literature. CASTEP is dual-licensed: full source code is available world-wide for academic use under a cost-free license, and a paid-for commercial license is available from BIOVIA.

CASTEP was written in modern Fortran, using Message Passing Interface (MPI) to enable distributed-memory parallelism. Open multiprocessing (OpenMP) threading was added in 2014–2015, to reduce CASTEP's memory usage and improve parallel scaling on multicore architectures, and CASTEP simulations using OpenMP-MPI demonstrate excellent scaling across a range of simulations and computer hardware. However, the advent of exascale computing, in particular, the move toward heterogeneous computing, presents a significant challenge.

TOWARD EXASCALE COMPUTING WITH ACCELERATORS

Delivering exascale computing is challenging, and the majority of exascale machines are anticipated to have heterogeneous architectures. These machines will typically have a conventional CPU to manage the overall system, including the operating system and input/output (I/O), but a substantial fraction of the computational power will be delivered by accelerators.

At present, accelerators generally fall into the following two categories: field-programmable gate arrays and general-purpose graphical processing units (GPU). Of these two classes, GPU-based accelerators are the most widely available, and have the most mature software development frameworks. There are the following two attractive features of GPUs for large-scale high-performance computing: computational power and memory bandwidth. At the time of writing, an AMD Rome CPU can deliver around 2.5 TFLOPs in double-precision and 200 GB/s memory bandwidth, so a typical two-socket cluster node can achieve 4 TFLOPs from the CPU and 400 GB/s total memory bandwidth. In contrast, a single NVIDIA V100 GPU can deliver almost eight TFLOPs in double-precision and an A100 GPU can deliver nearly 10 TFLOPs, with on-card memory bandwidths of 0.9 and up to 2 TB/s, respectively. Node architectures are available which feature eight GPUs per node, yielding almost 80 TFLOPs and 16 TB/s per node—over 20 times the computational power and memory bandwidth of the two CPUs.

The high memory bandwidth of GPUs is delivered via HBM2 on the GPUs themselves; when data is being moved from system RAM, the memory bandwidth is throttled by the link between the GPU and the motherboard. On a PCI Express 4.0 link the maximum bandwidth is 64 GB/s, and even the promised 128 GB/s from PCIe 5.0 is considerably less than the CPU memory bandwidth, let alone that of the GPU. This figure improves with specialist communication links such as NVIDIA's NVLink, which can deliver 600 GB/s; nevertheless, data movement remains relatively slow and can easily become a bottleneck in scientific computation.

As GPU hardware has evolved to become a more general-purpose computational resource, new software frameworks have been developed to provide more hardware abstraction and performance. For Fortran-based software such as CASTEP, there are essentially three main choices of software technology: CUDA Fortran, OpenMP (from 4.0 onward), and open accelerator (OpenACC). CUDA is a parallel programming framework developed by NVIDIA for the specific purpose of running computational software on GPUs. CUDA itself is essentially a proprietary extension to C/C++ to enable GPU calculations and manage memory between the host (CPU) and device (GPU); CUDA Fortran is the corresponding extension to the Fortran language. CUDA Fortran enables fine-grained control of the data movement, memory management, and calculation details of computational kernels, but it is also a proprietary extension supported only by NVIDIA's Fortran compiler, and can only target NVIDIA GPU hardware.

OpenMP is a directives-based programming framework, which was originally developed as a shared-memory parallel framework to exploit multicore CPUs. It is well-supported by a wide range of C, C++, and Fortran compilers, including commercial compilers from Intel, Cray, and NVIDIA, as well as Open Source compilers such as the GNU Compiler Collection (GCC). Support for offloading OpenMP regions to accelerators was only introduced in OpenMP 4.0, and this has subsequently been extended (first in OpenMP 4.5 and then further in OpenMP 5).

OpenACC is also a directives-based programming framework but, unlike OpenMP, is specifically designed to enable computation to be offloaded to accelerators. OpenACC-enabled compilers are available for C, C++, and Fortran, including NVIDIA's compilers. Limited support for OpenACC was also introduced in the Open Source GCC compiler suite with version 5.1, and was developed steadily over successive versions; however, it was only with the advent of GCC 10 that a sufficient subset of OpenACC was available for large research software to use it efficiently.

In this work, OpenACC is the software framework of choice, principally because it is a mature, open standard for offloading computational work to accelerators and is not tied to a particular compiler suite or hardware vendor. The same may hold for OpenMP in the future, but at present it is neither as fully featured nor as lightweight as OpenACC.

CASTEP'S COMPUTATIONAL KERNELS

The main workload in a CASTEP calculation is the application of the Hamiltonian matrix, as described earlier. The kinetic energy matrix is trivial to apply in Fourier space, and takes negligible time in most practical calculations. In contrast, the local potential matrix is most efficiently applied in direct space, and thus, every band b at wavevector $\mathbf{k}$ requires a pair of inverse and forward fast Fourier transforms (FFTs); the inverse FFT transforms ψ_{bk} into direct space, permitting efficient application of the potential V_{loc}, and the forward FFT returns ψ_{bk} to its native Fourier space. The standard 3-D FFT methods scale favorably with simulation size (for a single band at a single $\mathbf{k}$-point), but have a relatively large prefactor and so take a significant amount of computational time for small- and medium-sized simulations. This is in part because the compute intensity is low for FFTs, leading to the algorithm being memory-bound in many cases, and also because each application of the Hamiltonian in plane-wave DFT requires two FFTs for every band at every $\mathbf{k}$-point. GPUs typically benefit from extremely high memory-bandwidth, which means that FFTs can be much quicker when compared to the CPU performance once the data has been offloaded to the GPU. The data transfer itself, however, can be a bottleneck, precisely because the compute intensity is low, and care must be taken both to maximize data locality with respect to the CPU or GPU, and also to maximize the data operations performed on the GPU.

The nonlocal pseudopotential matrix is a low-rank matrix update, which may be applied efficiently using standard linear algebra packages (e.g., OpenBLAS). The scaling of the matrix with system size is linear in the number of updates and the size of each update, and is consequently quadratic in the overall storage requirements and cubic in computational cost with regards to a straightforward application. This cubic scaling means that applying V_{nl} becomes a significant fraction of the total computational time for large simulations. A real-space truncation method can reduce this to linear scaling in memory and quadratic scaling in time, but reduces accuracy to a level which is unacceptable in many research applications. However, this operation is entirely composed of cache-friendly matrix–matrix operations, for which GPUs excel. Furthermore, for large simulations, the quadratic scaling of the data volume renders the CPU-to-GPU (or GPU-to-CPU) transfer time negligible compared with the cubically scaling workload.

CASTEP GPU PORT

The work presented here is based on the CASTEP 18.1 codebase, augmented with OpenACC directives to control the allocation, deallocation, and transfer of data structures, generation of GPU kernels and interfacing to optimized libraries (cuFFT for FFTs and cuBLAS for linear algebra). As far as possible, the directives were restricted to CASTEP's low-level "utility" modules, with an additional "accelerator" module to handle meta-data and control logic specific to the OpenACC code-paths. For performance reasons, some OpenACC directives were introduced into CASTEP's mid-level "fundamental" modules, which define operations on, for example, wavefunctions, densities and potentials, and a small number in higher level "Functional" modules (principally those operations involving nonlocal potentials). These additions are comparable in scope to the existing OpenMP directives for threading on CPUs, and in many cases are in the same areas of the codebase.

GPU Performance

The performance of the GPU port was tested on the Bede Tier-2 HPC facility. Bede is an IBM Power9-based system, with 32 GPU-based nodes, as well as 4 "inference" nodes. Each of the GPU nodes has 32 Power9 cores and 4 NVIDIA V100 GPUs, each with NVLink 2.0. Initial tests focused on the single-GPU performance, using 8 MPI processes with and without a single V100 (shared between the processes), and on small CASTEP simulations to enable rapid development-test-optimize cycles. These initial results showed that offloading only the nonlocal potential to the GPU gave a small speed-up of x1.1, whereas offloading the local potential gave a speed-up of x1.7; combining the two gave a speed-up of over x1.8. Further optimization to reduce the density calculation time, data movement, and extend the offloaded operations led to a speed-up of x1.95.

The benchmark simulation was chosen from an active research project investigating the effect of disorder in the Heusler alloy Fe_2VAl. This system was initially constructed from 12 primitive cells, and has 24 Fe atoms, 12 V atoms, and 12 Al atoms. The Brillouin zone was sampled with a $6 \times 6 \times 5$ Monkhorst-Pack grid, giving 90 $\mathbf{k}$-points in the symmetrized set. CASTEP's on-the-fly ultrasoft pseudopotentials were used, with a well-converged plane-wave cut-off energy of 800 eV. The semilocal PBE functional was used as the exchange–correlation functional.

Parallelization Strategies

Exascale computing will of course not be achieved with a single accelerator, and it is important that any GPU port is able to use multiple CPUs and GPUs efficiently. The most efficient way to exploit multiple cores in CPU-only calculations is by distributing the data and workload over the $\mathbf{k}$-points. Since the construction and application of the Hamiltonian matrices may be performed almost independently at different $\mathbf{k}$-points, this was also expected to be an efficient use of the GPU port. Figure 1 shows the time, performance, and parallel scaling of the GPU-port from 8 to 90 CPU cores of Bede (1 to 12 GPUs). In these calculations, up to 8 MPI processes share each GPU, and all the results presented used NVIDIA's CUDA multiprocess service, which allows concurrent execution of kernels from different processes. The GPU port shows a good performance across the whole range of CPU and GPU counts, consistently achieving approximately x2 the performance of the CPU-only calculation and scaling well with increased numbers of CPUs and GPUs.

The benchmark calculation has 90 $\mathbf{k}$-points, which are divided as uniformly as possible amongst the MPI tasks. Where a uniform division is not possible, there will be a load imbalance between different MPI tasks, with some tasks having one more $\mathbf{k}$-point than the others. Whenever communication is required between the tasks, those tasks with fewer $\mathbf{k}$-points will arrive at the communication point first and must inevitably wait for the more heavily loaded tasks to catch up. This effect is negligible for small numbers of MPI tasks, where a single $\mathbf{k}$-point is only a modest fraction of the total assigned to each MPI task, but it becomes significant as the number of MPI tasks increases, and the corresponding number of $\mathbf{k}$-points per task decreases. For large numbers of tasks, such load imbalance degrades the parallel performance severely and can completely skew the apparent parallel efficiency. For this reason, we have restricted our larger parallel calculations to the load-balanced choices of 30, 45, and 90 MPI tasks, corresponding to 3 $\mathbf{k}$-points, 2 $\mathbf{k}$-points, and 1 $\mathbf{k}$-point per MPI task, respectively.

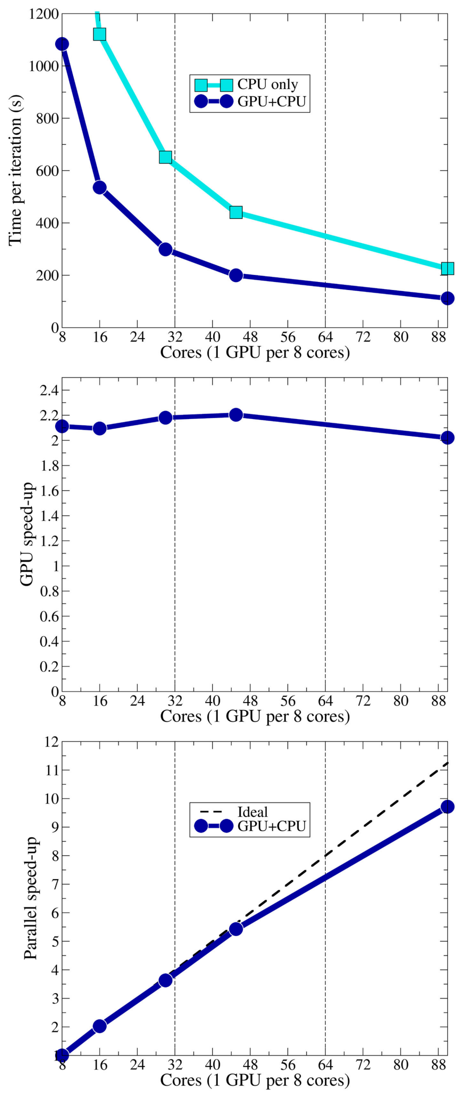

FIGURE 1. Performance of the CASTEP GPU port on Bede, for a research simulation of disorder in the Heusler alloy Fe_2VAl. Core counts are chosen to give good load balancing, as discussed in the text. (Top) Time per iteration for both CPU-only and CPU+GPU simulations; (Middle) The performance of the GPU port compared to CPU-only; (Bottom) The parallel scaling of the CPU+GPU simulations, using 1 GPU per 8 CPU cores. Vertical dashed lines indicate node boundaries.

For a simulation using 90 $\mathbf{k}$-points, it is clear that no more than 90 MPI tasks may be used when running with $\mathbf{k}$-point parallelism only. In CPU-only CASTEP simulations, additional cores would usually be exploited by distributing the Fourier coefficients for each $\mathbf{k}$-point. On 180 CPU cores, for example, each pair of cores would share the Fourier coefficients for a single $\mathbf{k}$-point.

This distribution is efficient for most of the computational operations, but the distribution of the Fourier data means that the 3-D FFTs can no longer be performed by a single process; each 3-D FFT is instead decomposed into three successive sets of 1-D FFTs (one set of FFTs along the Cartesian x-direction, followed by a set along y, followed by a set along z). A global data transposition is required between each FFT set, such that the data passed to the 1-D FFT library is local to the MPI process and contiguous in memory. For example, if the MPI tasks' local data is contiguous along the x-direction (with data at different y and z coordinates distributed over MPI tasks) then each task may perform its portion of the set of 1-D FFTs along x. The next stage of the method is to perform the set of 1-D FFTs along y, but these data are distributed across the MPI tasks, and even the local portion of the data is contiguous in x, not y. The global data transposition exchanges and reorders the data between all relevant MPI tasks such that the set of 1-D FFTs along y may be performed; a similar data transposition is then performed in preparation for the set of 1-D FFTs along z.

THE DISTRIBUTION OF THE FOURIER DATA MEANS THAT THE 3-D FFTs CAN NO LONGER BE PERFORMED BY A SINGLE PROCESS.

This parallel FFT algorithm poses several challenges to the optimization of the GPU port. 1-D FFTs have a lower compute intensity than 3-D transforms, so offloading these operations to the GPU intrinsically yields a lower speed-up than obtained in the 3-D case. Distributing the 1-D FFTs also reduces the volume of data for each GPU, and consequently the GPU performance for large parallel calculations is more susceptible to latency in both the data transfer and the launching of the computational kernels on the GPU. (In fact, kernel launch latency can also be a significant factor for 3-D FFTs, as will be seen in a later section.)

The global data transpositions, which are required in between each set of FFTs, involve an all-to-all communication which exchanges and reorders data amongst all of the participating MPI tasks. In the current CASTEP GPU port, all communications are handled by the MPI tasks on the CPUs, which introduces additional GPU-to-CPU and CPU-to-GPU data transfers when the FFTs are offloaded to GPUs. Use of a "CUDA-aware" MPI environment would ameliorate this problem by allowing the MPI library to send and receive GPU data directly, circumventing the intermediate data copies to and from the CPUs. Unfortunately, the packaging of the wavefunction data into the MPI buffers requires not only nonunit-stride data accesses, which are inefficient on GPUs, but also a separate OpenACC kernel to be launched *for each remote process*. As the number of participating MPI processes in a calculation increases, so too does the number of kernels launched, but the workload per kernel decreases. This simultaneously increases the kernel launch latency and reduces the vector efficiency of each kernel.

One alternative parallel strategy is to keep 90 MPI tasks, but exploit additional CPU resources using OpenMP threads. In this approach, each MPI task has a number of OpenMP threads associated with it and, in order to exploit this fully, each subroutine in CASTEP must be threaded using OpenMP directives. OpenMP is a shared-memory parallel paradigm, which avoids the need for the Fourier coefficients to be distributed, since each thread has access to the whole dataset. In this mode of parallel operation, the Fourier transforms remain full 3-D FFTs, and the excellent GPU performance is retained; CPU-only calculations may instead use threaded 3-D FFT libraries.

The memory associated with an MPI task is shared amongst all of its OpenMP threads, and this presents an opportunity for operations to be parallelized over threads in ways which would not be possible for the entire calculation. In particular, when applying the nonlocal potential V_{nl}, of (4), the operation may be threaded over the matrix updates themselves, rather than the wavefunction to which they are applied. This gives an efficient way to parallelize the application of V_{nl}, which is independent of any other parallelization methods. The present OpenACC port contains some regions, which are only threaded in the non-OpenACC code paths, meaning that CPU-only calculations gain more benefit from threading than calculations using GPUs. Nevertheless, running the benchmark calculation on 180 cores via 90 MPI tasks, and 2 OpenMP threads per task, the GPU version still achieves a speed-up of over x1.6, with an iteration time of 107 s compared to 174.3 s for the CPU-only simulations.

NLXC PERFORMANCE

In the preceding section, much of the acceleration of CASTEP when using the GPUs was driven by the performance improvement of the 3-D FFTs in the application of the local potential. Because the local potential requires two Fourier transform for every one of the N_b bands at N_k $\mathbf{k}$-points, the total computational cost scales as $N_\mathrm{b}N_\mathrm{k}N\log N$, where N is the size of a single FFT.

A well-known problem with semilocal exchange–correlation functionals is that the energies of localized states, such as those of d- and f-electrons, are often too high, compared to other states. This incorrect energy can lead to the simulation predicting the wrong electronic states being occupied, and can change the chemical bonding, leading to inaccurate simulations of important material properties, such as electrical conductivity or whether the material is magnetic.

The main cause of this error can be avoided by using the aforementioned class of NLXC approximations. In these methods, the extra term in the Hamiltonian, V_nlxc, is constructed from the pairwise density

$$\rho_{\mathrm{b}\mathbf{k}\mathrm{b}'\mathbf{k}'}(\mathbf{r}) = f_{\mathrm{b}\mathbf{k}}\psi_{\mathrm{b}\mathbf{k}}(\mathbf{r})\psi^*_{\mathrm{b}'\mathbf{k}'}(\mathbf{r}). \qquad (5)$$

The construction and application of V_nlxc requires two FFTs per *pair* of bands and $\mathbf{k}$-points; i.e., the total computational cost scales as $N_\mathrm{b}^2N_\mathrm{k}^2N\log N$. This is a substantial increase in the computational workload compared to semilocal exchange–correlation, and usually means that the computational time is dominated by the FFTs even for relatively large simulations. Indeed, it is common for FFTs to comprise over 95% of the total simulation time.

INDEED, IT IS COMMON FOR FFTs TO COMPRISE OVER 95% OF THE TOTAL SIMULATION TIME.

As was seen in the earlier section, the performance of 3-D FFTs is improved greatly on GPUs, and it is natural to expect that NLXC calculations should see a large performance improvement. Because much of the accelerator code was encapsulated in low-lying utility modules, extending CASTEP's GPU port to NLXC operations required relatively few explicit OpenACC clauses. Since the computational cost of NLXC calculations is much larger than that of semilocal simulations, a smaller test case was chosen for testing and benchmarking: a $2 \times 2 \times 2$ supercell of the primitive unit cell of Fe_2VAl, consisting of eight formula units (i.e., 32 atoms in total). The simulations were performed on another Power9-V100 machine, the Ascent system at Oak Ridge National Laboratory. Each node of the Ascent machine has two 22-core Power9 CPUs (only 21 cores are available for computation on each CPU) and 6 V100 GPUs, with an NVLink 2.0 interconnect. The Power9 processor cores have a theoretical peak double precision performance of 24.6 GFLOPs each, and a memory bandwidth of 21.3 GB/s per channel (each socket having eight channels). Considering the 42 available compute cores per node, this leads to a theoretical per-node CPU performance of 1.0 TFLOPs and a memory bandwidth of 0.34 TB/s, compared to 46.8 TFLOPs and 5.4 TB/s for the 6 V100 s combined.

TABLE 1. Analysis of the calls to the CUDA API with the initial CASTEP-GPU port of NLXC.

CUDA operation	Time (s)	Proportion of API calls
LaunchKernel	50.8	61%
StreamSynchronise	26.0	31%

The simulation is for the bulk Fe_2VAl system described in the text.

The performance benchmarking was carried out on a single node of Ascent, using two MPI tasks (one per CPU socket). For the CPU-only simulations, each MPI task used 21 OpenMP threads; for the GPU-enabled simulations, each MPI task was pinned to a different GPU, so in this work only two GPUs were utilized. The average iteration time for the CPU-only simulation was approximately 642 s; the initial GPU port of NLXC completed the same calculation with an iteration time of 80 s, a speed-up of x8. This performance improvement over the CPU-only calculation is approximately four times that achieved for the simulations using semilocal exchange–correlation methods, confirming the impressive performance of 3-D FFTs on GPUs. Nevertheless, detailed timeline and performance analysis of the simulations showed that the GPUs were underutilized, and an analysis of the CUDA calls generated at run-time showed that over 90% of all calls to the CUDA API were launching compute kernels and waiting to synchronize kernels and data transfers (see Table 1). These operations took a sizeable amount of time, and the detailed profile highlighted that the computational workload itself was accelerated considerably more than the observed x8 speed-up.

Most of the kernels launched were forward- and inverse FFTs, as expected, and the large number of kernels was due to the NLXC operations being performed on a single pair of bands at a time. Refactoring the code to work on blocks of bands enabled the FFTs to be carried out in batches, reducing the number of kernels launched and simultaneously increasing the computational work per kernel, which itself allows greater GPU utilization. The refactoring also extended the OpenACC regions to fuse OpenACC kernels, further reducing the

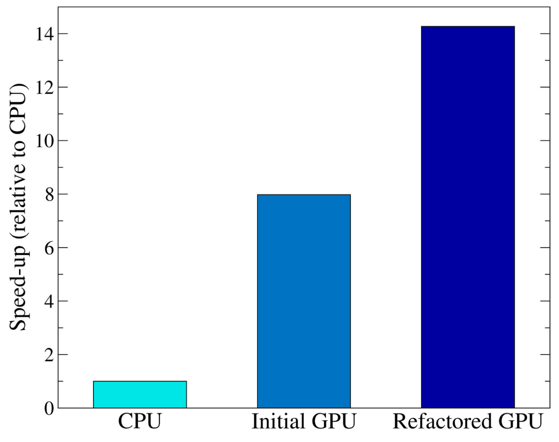

FIGURE 2. Performance of the initial and optimized GPU ports, compared to a pure CPU calculation. The benchmark was a "screened-exchange" NLXC simulation of bulk Fe_2VAl. All simulations were performed on the Ascent machine at ORNL, using 2 MPI tasks; each task used either 21 CPU threads or 1 GPU.

number of kernels launched and increasing the work per kernel, as well as optimizing data movement.

Benchmark simulations of the new code showed that the refactoring had improved the GPU performance substantially, further reducing the iteration time by over 40% to 45 s; thus, the overall speed-up was over x14 compared to the baseline CPU-only simulations (see Figure 2). This speed-up is close to the theoretically ideal speed-up of x15.8, calculated from the ratio of the per-node GPU and CPU memory bandwidths.

CHALLENGES AND FURTHER WORK

Although OpenACC has many advantages for widely used, portable software such as CASTEP over proprietary software technologies, there remain a number of challenges to be tackled. A major reason behind the decision to adopt OpenACC for the porting was that it is an open standard, which is relatively hardware-agnostic, and capable of offloading calculations to different accelerator architectures. In practice, however, few Fortran compilers have an OpenACC implementation, and the work here was carried out using the NVIDIA Fortran compiler (which will only target NVIDIA GPU accelerators).

OpenACC was designed to enable "off-loading" of work from a CPU to a GPU, and this is achieved by choosing any relevant CPU data objects and mapping them to corresponding data objects on the accelerator. However, when entire operations are off-loaded to the GPU there are occasions when some objects, for example, temporary arrays, are *only* required on the *accelerator*. The OpenACC "device_resident" attribute is designed for exactly this use-case, but at present there are no functioning implementations of it. There is also an acc_malloc routine for exactly this situation, but it is only supported in C/C++, not Fortran. Similarly, the OpenACC 3.0 specification supports the mapping of CPU pointers to GPU addresses (and vice versa), and thereby facilitates the use of memory pools to avoid reallocating memory unnecessarily, but this is also only available in C/C++. This means that, at present, dummy arrays must be created on the CPU, and the data mapped between the CPU and accelerator objects. This increases data copies, which are already the bottleneck in many off-load operations, as well as the memory footprint of the CPU.

In an earlier section, the hybrid OpenMP-MPI approach was discussed as a way to parallelize GPU-CASTEP calculations beyond the straightforward $\mathbf{k}$-point parallelism. An alternative approach is to distribute the data and workload by the bands (index b) of the wavefunction, as well as the $\mathbf{k}$-points. The 3-D FFTs are performed on each band independently, so this decomposition retains the ability for each MPI task to offload entire 3-D FFTs to the GPUs. The Hamiltonian matrix only depends on the $\mathbf{k}$-point, not the band index, so the same Hamiltonian is applied to each of the bands at the same $\mathbf{k}$-point. In the current GPU implementation, several MPI tasks share a GPU and each MPI task transfers the Hamiltonian data to the GPU along with its portion of the wavefunction. In band-parallel calculations, this approach leads to a substantial inefficiency because the Hamiltonian data *is the same* for all the MPI tasks working on the same $\mathbf{k}$-point. Thus, the MPI tasks which share a GPU all send identical copies of the data to the GPU. A much more efficient approach would be for one MPI task to send the data to the GPU, and for all of the MPI tasks to share that data. Such sharing of GPU data between MPI tasks is possible using CUDA and IPC, but a pure OpenACC implementation is not straightforward and is still under development.

Finally, for massively parallel calculations it is desirable to be able to use CASTEP's Fourier-parallelism efficiently with GPUs, in addition to the other parallel methods. The excellent performance of the NLXC simulations suggests that some of the shortcomings of Fourier-parallelism could be addressed by batching the 1-D FFTs, which results in fewer, larger data transfers and GPU kernels; unfortunately, this is hindered by a shortcoming in the current implementations of batched FFTs. Each of CASTEP's 1-D FFTs are performed in-place, regardless of whether they are along the x-, y-, and z-directions. In the general case, the length of the transformations is not the same in each of these directions, and so CASTEP's data buffers are allocated for the maximum size required, and shorter transform data is padded with zeroes. Whilst the single 1-D FFT subroutines handle this with ease, the current batched FFT subroutines do not, requiring instead that the data for the next FFT starts *immediately* following the previous FFT data. Repacking the data into a separate, contiguous buffer would increase the memory footprint and, more importantly, introduce another set of short, relatively inefficient kernels. This effectively precludes the use of batched 1-D FFTs in CASTEP at present.

CONCLUSION

We have presented the work-in-progress port of the CASTEP first-principles materials modeling program to accelerators. CASTEP is written in modern Fortran, and the port used OpenACC directives to offload several key computational kernels to accelerators. The work demonstrates that OpenACC is a viable route to porting large research software to accelerators. Nevertheless, several specific issues have been identified, in particular shortcomings in the present implementations, and situations where features available to C/C++ programs are not available in modern Fortran.

Benchmarking on two Power9-based NVIDIA Volta GPU systems shows a speed-up of x2 when using GPUs to accelerate standard simulations, and x14 for simulations using NLXC methods. GPU-based accelerators are anticipated to be key components in the first exascale HPC facilities, and this work also represents the first steps to adapting CASTEP to exploit such resources. The GPU port has been designed to be parallel from the outset, and we have already demonstrated excellent scaling up to a dozen GPUs, even for small simulation sizes; nevertheless, this falls far short of the tens or hundreds of thousands of GPUs which are likely to be required to deliver exascale computing. The key immediate challenges have been identified and discussed, as well as possible future approaches, but achieving the extreme level of GPU-parallelism required for exascale may well go beyond designing efficient parallel offload models, and require deep, novel algorithmic changes in the core of CASTEP itself.

ACKNOWLEDGMENTS

The authors would like to thank Mike Payne (Cambridge), who facilitated the early stages of this project, and has provided unfailing help, support, and encouragement throughout. Filippo Spiga (formerly Cambridge, now NVIDIA) has proven to be a font of GPU wisdom and has been most generous with his time, providing considerable help, advice, and guidance. Paul Calleja (Cambridge) has also supported this work with much-needed resources.

Ed Higgins (York) wrote much of CASTEP's original OpenMP code, and provided invaluable help with the CASTEP GPU development during the hackathons; he is involved in the on-going development of the CASTEP GPU port, which will be the subject of a future publication. The authors would also like to thank NVIDIA, in particular, Mark Berger (now retired), for encouraging this project with enthusiasm and arranging training, engineer time, and hardware resources to facilitate the software development underpinning this work, and the GPU hackathon teams, organizers, and mentors at the University of Sheffield, U.K., and Oak Ridge National Laboratory (ORNL), USA, in particular, Paul Richmond (Sheffield), David Applehans (formerly of IBM, now NVIDIA), and Alan Gray (NVIDIA).

This work made use of the facilities of the N8 Centre of Excellence in Computationally Intensive Research (N8 CIR) provided and funded by the N8 research partnership and EPSRC under Grant EP/T022167/1. The Centre is coordinated by the Universities of Durham, Manchester, and York. Some of the development, and all of the NLXC calculations, were performed on the Ascent and Summit systems of the Oak Ridge Leadership Computing Facility, ONRL, which is supported by the Office of Science of the U.S. Department of Energy under Contract DE-AC05-00OR22725; access to Summit was provided by a Director's Discretionary Grant CHP108, which was only possible with the support and assistance of Ada Sedova and Anibal Ramirez Cuesta (ORNL). Development work was also performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council under capital Grant EP/P020259/1.

Financial support for this work was provided by EPSRC through a Research Software Engineer Fellowship (EP/R025770/1), the EPSRC Centre for Doctoral Training in Computational Methods for Materials Science (EP/L015552/1), the Materials and Molecular Modelling Exascale Design & Development Working Group (EP/V001256/1) and Cambridge Enterprise.

REFERENCES

1. W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation," *Phys. Rev.*, vol. 140, 1965, Art. no. A1133, doi: 10.1103/PhysRev.140.A1133.
2. P. J. Hasnip, K. Refson, M. I. J. Probert, J. R. Yates, S. J. Clark, and C. J. Pickard, "Density functional theory in the solid state," *Phil. Trans. R. Soc. A*, vol. 372, 2014, Art. no. 20130270, doi: 10.1098/rsta.2013.0270.
3. R. O. Jones, "Density functional theory: Its origins, rise to prominence, and future," *Rev. Mod. Phys.*, vol. 87, 2015, Art. no. 897, doi: 10.1103/RevModPhys.87.897.
4. S. J. Clark *et al.*, "First principles methods using CASTEP," *Z. Kristallogr.*, vol. 220, pp. 567–570, 2005, doi: 10.1524/zkri.220.5.567.65075.

MATTHEW SMITH is currently a Research Software Engineer and a Ph.D. student. His work includes preparing first-principles materials modeling software for use on candidate exascale hardware; in addition to CASTEP's GPU port, his recent work includes optimizing CASTEP's 3-D FFTs for massively parallel architectures. Contact him at msmit1@bsc.es.

ARJEN TAMERUS is currently a High-Performance Computing Consultant with the University of Cambridge, Cambridge, U.K., where he extends, develops, and optimizes research software on a range of hardware platforms, including the CSD3 Tier-2 HPC service. He has worked on several large research software projects, such as the Square Kilometre Array, and in addition to contributing to CASTEP's OpenACC port, he has worked on a variety of node-level optimizations for CASTEP, including OpenMP. Contact him at at748@cam.ac.uk.

PHIL HASNIP is currently an EPSRC Research Software Engineer Fellow and a Principal Author of the CASTEP first-principles materials modeling software. He has led many of CASTEP's technical developments, including the implementation of OpenMP- and band-parallelism, and the OpenACC port. He is a co-investigator on the U.K. Car-Parrinello High-End Compute Consortium and the U.K. Materials and Molecular Modeling Exascale Design and Development Working Group, and was a member of the Benchmarking Committee for the U.K. Tier-1 ARCHER2 procurement. Contact him at phil.hasnip@york.ac.uk.