CP2K Performance from Cray XT3 to XC30

Iain Bethune, Fiona Reid
EPCC, The University of Edinburgh
Edinburgh, UK
Email: {ibethune,fiona}@epcc.ed.ac.uk

Alfio Lazzaro
Cray Switzerland
Lugano, Switzerland
Email: alazzaro@cray.com

***Abstract*—CP2K is a powerful open-source program for atomistic simulation using a range of methods including Classical potentials, Density Functional Theory based on the Gaussian and Plane Waves approach, and post-DFT methods. CP2K has been designed and optimised for large parallel HPC systems, including a mixed-mode MPI/OpenMP parallelisation, as well as CUDA kernels for particular types of calculations. Developed by an open-source collaboration including University of Zürich, ETH Zürich, EPCC and others, CP2K has been well tested on several generations of Cray supercomputers, beginning with the XT3 in 2006 at CSCS, through XT4, XT5, XT/XE6 and XK7, to Ivy-Bridge and Sandy-Bridge based XC30 systems in 2014. We present a systematic view of benchmark data spanning 9 years and 7 generations of the Cray architecture, and report on recent efforts to carry out comprehensive comparative benchmarking and performance analysis of CP2K on the XE6 and XC30 systems at EPCC. We also describe work to enable CP2K for accelerators, and show performance data from the XK7 and XC30 at CSCS.**

***Keywords*-CP2K; Benchmarking; Cray XT/XE/XC Series; Application Performance;**

I. Introduction

CP2K is a popular and capable open source program for atomistic simulation which has been developed and deployed on a range of Cray systems installed at CSCS and EPCC. In section II we describe the features of CP2K, and a short summary of developments which have improved performance and scalability. Section III provides a history of the Cray HPC systems which we have benchmarked CP2K performance on, and the results of a single benchmark are presented and discussed in section IV. We also report on a recent performance comparison exercise on the XE6 and XC30 platforms using a wider range of benchmark systems (Section V). Finally in section VI we discuss ongoing developments to CP2K to harness heterogeneous CPU/GPU architectures.

II. CP2K Overview

CP2K [1] is a powerful and scalable program for atomistic simulations of a wide range of systems, including condensed phase, molecular systems and complex interfaces. Developed since 2001 by an international collaboration, CP2K is freely available under the GNU General Public license from the CP2K web site [2]. Although written in Fortran 95, CP2K is designed from the outset in an object oriented manner to allow easy extensibility and composability of different methods and algorithms. As a result CP2K features a wide range of force evaluation models including classical potentials, Semi-empirical schemes, Density Functional Theory, hybrid DFT-Hartree-Fock and post-HF correlation methods MP2 and RPA, as well as allowing arbitrary combinations of these. Built on top of these are many tools including Molecular Dynamics in various ensembles, Monte Carlo, geometry and cell optimisation, nudged elastic band, and Metadynamics (with PLUMED [3]). CP2K consists of around a million lines of code, and with an average of two commits per day to the SVN repository, development is rapid. To support this a set of over 2400 test input files is used as an automated regression test suite to ensure continued code correctness(see [4] for details), and also to provide examples for the growing user community.

The most widely known feature of CP2K is the QUICKSTEP [5] implementation of Density Functional Theory. QUICKSTEP adopts a dual basis approach to solving the Kohn-Sham equations, where atomic-centred Gaussian basis functionals are used to represent the wave functions, and an auxiliary basis of Plane Waves is used to expand the electronic density and efficiently compute the Hartree energy. The algorithm for transforming between the Gaussian basis stored as coefficients in a sparse matrix and Plane Waves stored on a regular 3D grid makes use of auxiliary 3D real space grids as a means to store the density before the Plane Wave coefficients are calculated using a Fast Fourier Transform. The mapping from matrix elements to the real space grids is referred to a *collocation* and the reverse as *integration*. As a result, the Kohn-Sham matrix (and total energy) can be computed in quasi-linear time - the FFT is $\mathcal{O}(nlogn)$ - and so can easily scale to thousands of electrons. In addition to QUICKSTEP, CP2K also implements the Orbital Transformation [6] method, as an alternative to the traditional diagonalisation approach to wavefunction orthogonalisation. While still cubically-scaling, OT has been demonstrated to outperform diagonalisation by a factor of 10 or more for typical systems. The combination of these two approaches gives CP2K excellent efficiency and the ability to simulate large systems within the local DFT approximation. Efficient parallelisation of these algorithms on contemporary HPC systems has taken place over a number of years, and

we now describe the key developments.

A. Parallelisation and Optimisation

Many improvements to CP2K have been made incrementally over the years, nevertheless we highlight here a small number of significant changes which have resulted in significant performance improvements for QUICKSTEP calculations.

- **Distributed Realspace grids**. The realspace grids used to transfer data between Gaussian and Planewave representations were originally implemented as replicated data multi-grids. For typical calculations, the finest grids might have a few hundred grid points on each side, and the coarsest a few tens of points, although calculations with large unit cells and/or high planewave cutoffs could be much larger. A multi-grid scheme is used so that diffuse gaussians can be mapped to coarse grids and highly peaked gaussian to the finer grids, supporting each gaussian with approximately the same number of grid points. Thus good accuracy can be obtained for minimal computational effort. In 2007 Slater and Watkins [7] implemented a domain decomposition scheme for the realspace grids, which allows a 1D, 2D or 3D geometric partitioning to be used depending on the number of processors. For very large grids, this overcomes the memory bottleneck associated with replicated data, but also overcomes communication costs associated with ensuring every MPI process has an up-to-date copy of the entire grid. Since the spatial distribution of basis functions is system-dependent and typically non-uniform, a load balancing scheme is employed to distribute the task of mapping each gaussian to/from the grids as evenly as possible, subject to the spatial constraints of the decomposition. Once load balancing is completed at each grid level, the assignment of grid sections to processes on subsequent levels may be re-ordered to pair lightly loaded sections with heavily loaded sections and reduce the maximum total load per process. Finally, the coarsest grid level(s) are usually replicated since they a cheap in memory and communication costs, and the replicated tasks are used to complete the load balance. For full details of the scheme see [8].
- **Fast Fourier Transforms**. Distributed FFT can often be a bottleneck in DFT calculations due to the need for `MPI_Alltoall` operations to globally transpose data (see [9] for an overview of the issues). CP2K originally supported a 1D (slab) decomposition for the Planewave grids, which at the time was enough to support MPI parallelism on 100s of processes. However, to enable scaling beyond this point, a 2D decomposition and corresponding two-stage transpose was implemented by VandeVondele and Hutter in 2008. Although CP2K includes a quite efficient set of hard-coded FFT routines, these are typically only used as a fall-back in case an optimised FFT library is not available. CP2K has a library-independent FFT interface, which can be implemented by FFTW3 [10] (including MKL via its FFTW3-compliant interface) or CuFFT if a CUDA-capable GPU is present). Additional performance tuning of FFTW3 by specifying the FFTW plan style are exposed to the user as options in the CP2K input file, and plans are cached in an FFT scratch data structure and may be re-used throughout a calculation.
- **OpenMP**. While the existing MPI implementation was designed during the era of 'traditional' MPP systems in the early 2000s, the need to scale further and the development of multi-core systems with more and more cores per node meant that a mixed-mode parallelisation scheme was needed. Some OpenMP support existed from the very early days of CP2K development, when the code was run on multi-processor IBM systems (e.g. the eServer p575, with 32 processors in a shared memory node), but this was modified, corrected in some places, and extended to cover all of the core QUICKSTEP operations [11]. While some parts of the code are still not fully OpenMP parallel, and this can be seen by the fact that the mixed-mode code does not perform as well as pure MPI for runs with relatively few cores, it offers the possibility to extend scalability and performance well beyond what can be achieved by MPI alone, and many of our benchmark results make use of this.
- **DBCSR**. Aside from regular grids, the other major data structures in CP2K are matrices with typical occupancies of 20-100%. Initially, a block-cyclic decomposition was used for compatibility with ScaLAPACK which handled (dense) linear algebra operations. However, the sparsity of the matrix was not exploited in communication, and multiplication of matrices was handled by bespoke routines which could become a bottleneck for large systems where OT dominates. As a result, a new sparse matrix library DBCSR (Distributed Block Compressed Sparse Row) was developed by Borsnik *et al.* [12] starting in 2009, which takes full advantage of the block-structured sparse nature of the matrices for efficient computation and communication. An auto-tuned library for small block DGEMMs (`libsmm`) is provided and has been shown to out perform vendor BLAS by up to 10x for particular sizes (see [13]). Indeed, a similar feature has now been implemented in Cray Libsci. A cache-oblivious recursive algorithm is used for local multiplication, and the parallelisation scheme is based on Cannon's algorithm, to ensure communication scales as $\mathcal{O}(logP)$ [14]. The DBCSR library has been recently ported to run on hybrid systems equipped with GPUs, showing good performance [15].

Table I
CRAY SYSTEM SPECIFICATIONS AT EPCC AND CSCS

Name	Arch.	Processor	Clock (GHz)	Nodes	Cores/ Node	Peak TFlop/s	GFlop/s/ Node	Year
XT3 Stage 0	XT3	AMD Opteron 146	2.0	84	1	0.336	4.0	2005
XT3 Stage 1	XT3	AMD Opteron 152	2.6	1100	1	5.72	5.2	2006
Piz Palü	XT3	AMD Opteron 185 Dual Core	2.6	1664	2	17.31	10.4	2007
HECToR Phase 1	XT4	AMD Opteron 1220 “Santa Ana” Dual Core	2.8	5664	2	63.44	11.2	2007
HECToR Phase 2a	XT4	AMD Opteron 2356 “Barcelona” 4-Core	2.3	5664	4	104.22	18.4	2009
Monte Rosa	XT5	AMD Opteron 2431 “Istanbul” 6-Core	2.4	1844	12	212.43	115.2	2009
HECToR Phase 2b	XT6	AMD Opteron 6172 “Magny-Cours” 12-Core	2.1	1856	24	374.17	201.6	2010
Piz Palü[1]	XE6	AMD Opteron 6272 “Interlagos” 16-Core	2.1	1496	32	402.12	268.8	2011
HECToR Phase 3	XE6	AMD Opteron 6276 “Interlagos” 16-Core	2.3	2816	32	829.03	294.4	2011
Tödi	XK7	AMD Opteron 6272 “Interlagos” 16-Core + NVIDIA Tesla K20X	2.1	272	16 (+14)	392.90	1444.5	2012
Piz Daint	XC30	Intel Xeon E5-2670 “Sandy-Bridge” 8-Core + NVIDIA Tesla K20X	2.6	5272	8 (+14)	7788.90	1477.4	2013
ARCHER	XC30	Intel Xeon E5-2697 v2 “Ivy-Bridge” 12-core	2.7	3008	24	1559.35	518.4	2013

The results will be discussed in section VI.

III. CRAY HPC SYSTEMS

CP2K has a long association with the Cray XT series of systems due to its popularity as both CSCS and EPCC. CP2K is the most widely used code at present on the CSCS systems Monte Rosa and Piz Daint, and usage has grown over the lifetime of the EPCC HECToR service from 4th to 2nd most popular code (measured by total CPU usage). In Table I we compile the specifications and performance of several generations of the Cray MPP architecture starting with the XT3 in 2006 through to the current day XC30. A number of trends are immediately apparent:

- **Clock speed and cores per node**. Barring the earliest XT3, there has been a steady decrease in CPU clock speeds, until the transition to Intel CPUs with the XC30. However, the number of cores per node has continued to increase. This corresponds to decreasing power utilisation of each individual core, allowing more cores to be packed into each compute node within a fixed thermal envelope. As discussed earlier, this trend one of the reasons why a mixed-mode MPI/OpenMP strategy has been implemented in CP2K.
- **Per-node performance**. These two factors, combined with increasing FLOPs per cycle - initially 2, then 4 on the Istanbul and later, and 8 on the Intel CPUs. Note that while AVX instructions were introduced in the AMD Interlagos (Bulldozer) core, the architecture shared a floating point unit between every two cores, keeping the peak FLOPs per cycle to 4. In the 9 years of data presented, the per-node performance has increased by a factor of 130 (or 370 if GPUs are included). This equates to a doubling of per-node performance at every 15 months. Clearly Moore’s law continues to hold, even the extra transistors come in the form of more cores rather than improved serial performance! To harness the extra performance available from vectorisation requires a combination of compiler, library, and source code support. In the case of CP2K, with a recent version of GNU `gfortran` or Intel `ifort`, a large percentage of the code can be vectorised directly by the compiler. For some operations, e.g. FFT, we impose particular alignment requirements on the arrays passed to the FFTW library to allow vector instructions to be used. Finally, some kernels such as the small matrix multiplications implemented in libsmm are generated with an vector length set at configure time to allow the compiler to vectorise for a given architecture.
- **AMD-Intel transition**. While the per-node performance increased with each new system, users were accustomed to finding their calculations running at a similar speed or even slower due to reductions in clock speed. This changed dramatically at the step from XE6 to XC30, where not only did clock speeds increase for the first time in years, but also the improved hardware AVX implementation resulted in a typical improvement of a factor of two or more (see section V for details). As mentioned before, the CP2K code was already modified to support AVX, so hardware transition enabled this improvement with negligible porting effort.
- **Network interface**. One detail not explicitly stated in Table I is the network architecture. The XT architectures all used a torus network, with the compute nodes attached by one of the SeaStar family of network adapters. The XE6 and XK7 use the Gemini router, and on the XC30, compute nodes are connected in a

[1]This system was subsequently renamed and is current known as Monte Rosa.

'Dragonfly' topology (essentially a fat-tree with all-to-all connectivity near the root of the tree), using the Aries router chip. While each generation of interconnect has provided increasing high-bandwidth and low-latency communications, these have typically not been the limiting factor for CP2K at the relatively modest processor counts discussed in section IV. The effects of the network were most obvious during the latter stages of the XT architecture, where the limited message injection rate (around 500,000 messages per second for SeaStar2+ compared with 4.5 million per second for Gemini [16]), combined with the rapidly increasing number of cores (and MPI ranks) per node, caused poor scalability of the FFT in CP2K. This effect could only be overcome by using OpenMP to reduce the number of MPI processes per node, resulting in fewer, larger messages traversing the network. However, the introduction of the XE6 and the Gemini router which was designed to cope with a wide multi-core architecture overcame the issue entirely. Furthermore, the Aries network improves the sustained injection bandwidth to 10 GB/sec (3 times that of Gemini), and also gives a large increase in global bandwidth (up to 20 times depending on configuration). This can greatly improve CP2K performance when running at very large scale.

IV. Molecular Dynamics of Liquid Water with Quickstep

Since early in the development of CP2K ab-initio molecular dynamics of liquid water using the Born-Oppenheimer approach has been used as a performance benchmark. It is easy to scale the system size by simply increasing the unit cell and adding additional water molecules and it provides a reasonable all-round test that is typical of many real-world applications involving condensed phases and reasonably small atoms. Production quality settings for the basis sets (TZV2P) and the planewave cutoff (280 Ry) are chosen, and the Local Density Approximation (LDA) is used for the calculation of the Exchange-Correlation energy. The initial configuration was generated by classical equilibriation, and the initial guess of the electronic density is made based on Atomic Orbitals. The smallest benchmark system - `H2O-32` - contains 32 water molecules (96 atoms, 256 electrons) in a 9.9 Å^3 cell, and the largest - `H2O-2048` - 2048 water molecules (6144 atoms, 49152 electrons) in a 39.5 Å^3 cell. These, as well as larger systems are available as part of the CP2K source distribution in `cp2k/tests/QS/benchmark`. In the following Figures 1-3, we plot the time per MD step against the number of CPU cores used. The diagonal dotted lines indicate perfect linear scaling. As most of the data reported is from machines which are now decommissioned, we are of course unable to fully separate the results of code changes and hardware effects by running newer versions of code on old systems, and vice versa. Thus, these results represent snapshots of the performance achievable on a given system at a given time with particular versions of code.

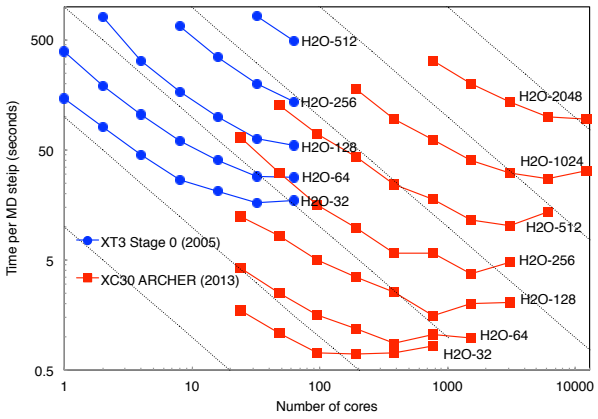

Figure 1. Performance of `H2O-32` up to `H2O-2048` benchmarks on Cray XT3 (2005) and Cray XC30 (2013)

In Figure 1 we show the measured time per MD step for benchmarks ranging from 32 up to 2048 water molecules on both the initial Cray XT3 system at CSCS from 2005 and the ARCHER XC30 system in 2013. For the smallest system sizes, where the scalability limit could be reached on the older machine, we observe an improvement in the peak performance of over 23 times (`H2O-32`) or 32 times (`H2O-64`). The performance achieved by a single compute node has increased by a factor of 84. This is around 65% of the improvement in the peak performance of the compute nodes, and this is due to the fact that on the XT3 a compute node has only 1 core, compared with 24 on ARCHER, and for such a small problem size scaling is not perfect even up to 24 cores. As well as the aforementioned increase in on-node performance, CP2K now scales to around 10 times as many cores due to improvements in network performance combined with better parallelisation (OpenMP, load balancing etc.). We note that calculations with large systems which were completely infeasible 10 years ago are now able to be performed routinely on today's systems.

Figure 2 presents a more detailed view of the 64 water molecule benchmark, showing performance on selected systems representing each architectural revision from XT3 through to XC30. Here we can clearly see the gradual improvements in scaling and performance from generation to generation, as well as the marked leap in performance from the AMD-based XT and XE systems to the Intel-based XC30. One interesting point to note is the relatively poor scaling of the code on the XT6 compared to the XT4 and XT5. This is due to the relatively under-powered SeaStar network adapter discussed in section III, and the XE6 with the Gemini network scales much better, despite having even more cores per node.

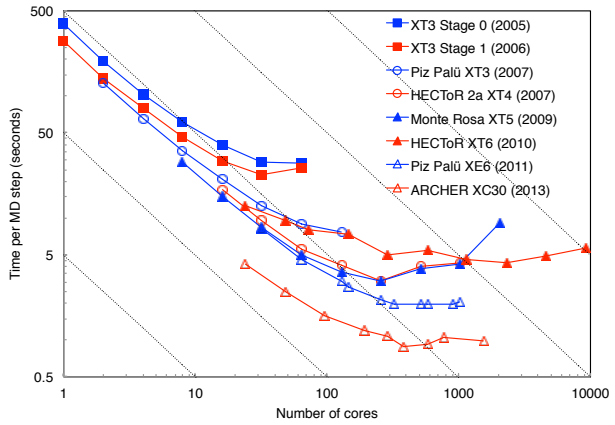

Figure 2. Performance of `H2O-64` on selected Cray systems

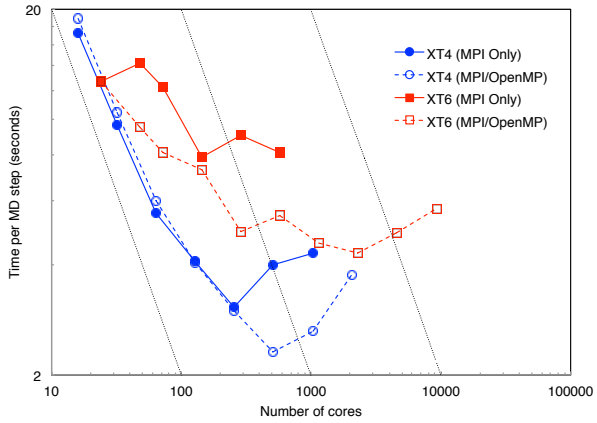

Figure 3. Performance of `H2O-64` on XT4 and XT6 with and without OpenMP

This effect is illustrated very clearly in the XT4 and XT6 results (Figure 3). On the XT4 with 4 cores per node, using MPI only gives the best performance up to the scalability limit at 256 cores. After this point, further scaling can be achieved up to 512 cores by using 4 OpenMP threads per process. By contrast, on the XT6 (24 cores per node), unless OpenMP is used the code can barely scale beyond a single node! Even so, the absolute run times are still slower than the XT4.

V. Comprehensive Benchmarking

While the `H2O-*` benchmarks provide a good view of CP2K performance for local DFT calculations over time, in recent years CP2K has been extended to includes much more accurate, but also more computationally demanding methods such as Hartree-Fock exchange [17], linear-scaling DFT [18], and the MP2 & RPA approaches [19] from many-body theory. A single benchmark system is no longer enough to give users useful information about what performance to expect from CP2K. We present a new benchmark suite which covers a wider range of CP2K functionality and show results comparing the EPCC HECToR Phase 3 (XE6) and ARCHER XC30 systems. We analyse the performance of CP2K for these test cases and explain the differences in performance observed between the two machines.

The aim of the benchmark suite is to provide a range of benchmarks which can be used to guide CP2K users towards the best configuration (e.g. machine, number of MPI processors, number of OpenMP threads) to use for their particular problem. The benchmark suite consists of five benchmarks; `H2O-64`, `Fayalite-FIST`, `LiH-HFX`, `H2O-DFT-LS` and `H2O-64-ri-mp2`. Descriptions of each benchmark and its performance are below. Since each some benchmarks perform MD, and the more expensive ones only a single-point energy computation, we report the total time for the calculation against the number of compute nodes used.

The mixed mode MPI/OpenMP version of the code is used to measure performance and we observed negligible overhead from running this version with 1 thread per process and the pure MPI code. For a fixed number of cores, all possible combinations of MPI processes and OpenMP threads were tested, subject to keeping each processes' threads within a single NUMA region. For example on HECToR Phase 3, 8 Interlagos cores share a single NUMA region, so we did not use more than 8 threads as the resulting performance would be very poor. As many HPC systems charge users by the node, full nodes are utilised at all times in our tests.

A. *H2O-64*

The `H2O-64` benchmark is taken from the standard water benchmark suite reported in section IV, and is included for consistency, even though it is now on the small side for simulations that would be routinely run on HECToR Phase 3 or ARCHER. It performs a short Born-Oppenheimer molecular dynamics run for 10 timesteps in an NVE ensemble at 300K. The system consists of 64 water molecules (192 atoms, 512 electrons) in a 12.4 $Å^3$ cell, using QUICKSTEP DFT with the LDA functional, a TZV2P basis set and 280 Ry cut-off.

Figure 4 shows the runtime of the `H2O-64` benchmark on ARCHER and HECToR Phase 3 plotted against the number of nodes. For each number of nodes we report only the best (i.e. fastest) of all the process/thread combinations tested. Above each data point the text gives the number of threads that gave the best performance where "MPI" means that using only a single thread was fastest. Below each data point the text gives the ratio of ARCHER/HECToR Phase 3 runtime. Comparing the performance of the two systems we can see that ARCHER is around twice as fast as HECToR Phase 3 and also scales better. For the H2O-64 benchmark we see that for smaller numbers of nodes the best performance is obtained using the MPI version of the code. For larger numbers of nodes the MPI scaling efficiency

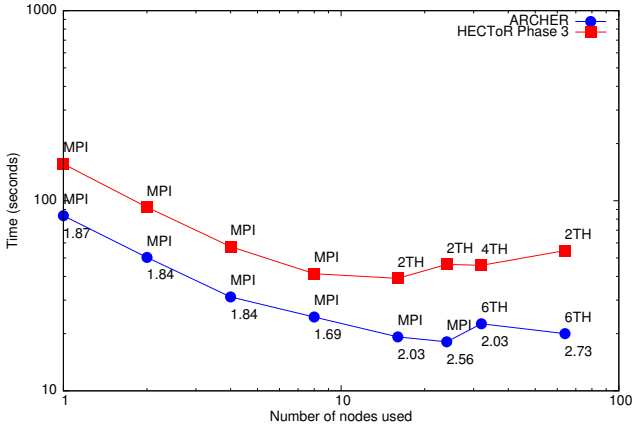

Figure 4. Performance comparison for the `H2O-64` benchmark plotting CP2K runtime against the number of nodes used. Above each data point the number of threads that gave the best performance is given. MPI means that the single thread (i.e. pure MPI) version of the code performed best. Below the ARCHER data points the ratio between ARCHER/HECToR Phase 3 runtimes is given. Thus a value of 2.03 means that the ARCHER runtime was 2.03 times faster than the HECToR Phase 3 runtime.

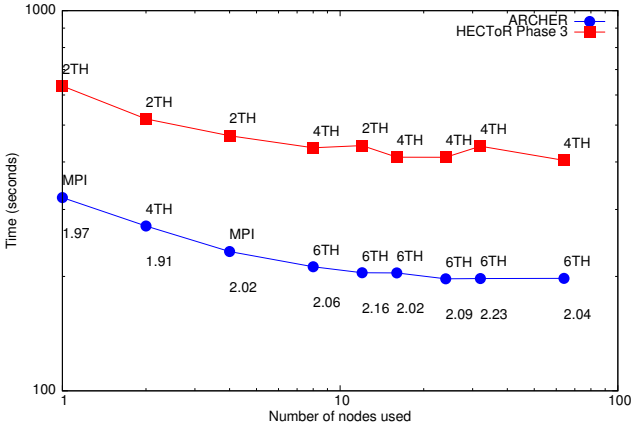

Figure 5. Performance comparison for the `Fayalite-FIST` benchmark.

reduces and using threads starts to be beneficial (e.g. when using more than 16 nodes on HECToR or 64 on ARCHER).

B. *Fayalite-FIST*

The `Fayalite-FIST` benchmark is another short molecular dynamics run of 1000 time steps in a NPT ensemble at 300K. It consists of 28000 atoms - a 10^3 supercell with 28 atoms of iron silicate (Fe_2SiO_4, also known as Fayalite) per unit cell. The simulation employs a classical potential (Morse with a hard-core repulsive term and 5.5 Å cutoff) with long-range electrostatics using Smoothed Particle Mesh Ewald (SPME) summation. While CP2K does support classical potentials via the Frontiers In Simulation Technology (FIST) module, this is not a typical calculation for CP2K but is included to give an impression of the performance difference between HECToR Phase 3 and ARCHER for the MM part of a QM/MM calculation.

The performance of the `Fayalite-FIST` benchmark is shown in Figure 5. Unlike the H2O-64 benchmark, for such a small (classical) problem, MPI scalability is limited and almost all node counts benefit from using threads. The difference between the best and worst runtime for a given number of nodes on ARCHER was typically 1-4% with 64 nodes taking up to 15% longer for the MPI version. The runtime is dominated by the short-range force calculation and I/O is also a contributing factor. The code is thus memory-bandwidth bound, and so using less MPI processes and giving each process more L3 cache is advantageous. As observed with the H2O-64 benchmark, ARCHER scales (slightly) better and is around two times faster than HECToR Phase 3.

C. *LiH-HFX*

The `LiH-HFX` benchmark is the first of three more demanding benchmarks and is a single-point energy calculation using Quickstep GAPW (Gaussian and Augmented Plane-Waves) with hybrid Hartree-Fock exchange. It consists of a 216 atom Lithium Hydride crystal with 432 electrons in a 12.3 $Å^3$ cell. These types of calculations are generally around one hundred times the computational cost of a standard local DFT calculation, although this can be reduced using the Auxiliary Density Matrix Method [20]. These calculations benefit particularly from using OpenMP as the HFX implementation requires a large amount of memory to store partial integrals. By using several threads, fewer MPI processes share the available memory on the node and thus enough memory is available to avoid recomputing any integrals on-the-fly, improving performance.

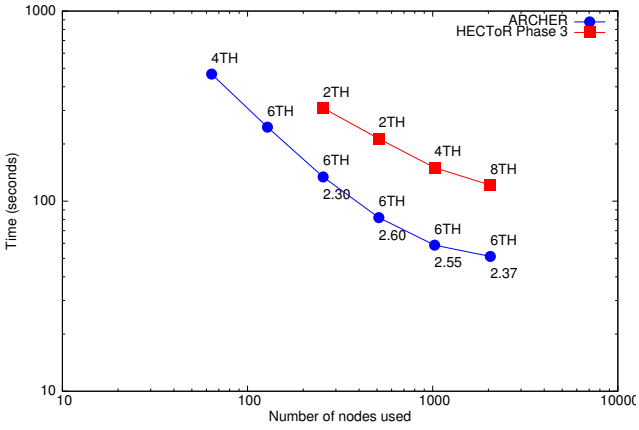

Figure 6. Performance comparison for the `LiH-HFX` benchmark.

The performance comparison of the `LiH-HFX` benchmark is given in Figure 6. No HECToR Phase 3 results are available for 64 or 128 nodes due to insufficient memory.

ARCHER has 64 GB memory per node compared with 32 GB per node on HECToR Phase 3 and thus it was possible to run the LiH-HFX benchmark on fewer nodes on ARCHER. In all our tests the best performance was obtained using at least 2 threads per process. As with the other benchmarks ARCHER generally scales better and runs at least 2.3 times faster than HECToR Phase 3.

D. H2O-DFT-LS

The H2O-DFT-LS (see Figure 7) benchmark is a single-point energy calculation using linear-scaling DFT. It consists of 6144 atoms in a 39 Å^3 box (2048 water molecules in total). An LDA functional is used with a DZVP MOLOPT basis set and a 300 Ry cutoff. For large systems the linear-scaling approach for solving Self-Consistent-Field equations will be much cheaper computationally than using standard DFT and allows scaling up to 1 million atoms for simple systems. The linear scaling cost results from the fact that the algorithm is based on an iteration on the density matrix. The cubically-scaling orthogonalisation step of standard QUICK-STEP DFT using OT is avoided and the key operation is sparse matrix-matrix multiplications, which have a number of nonzero entries that scale linearly with system size. These are implemented efficiently in the DBCSR library.

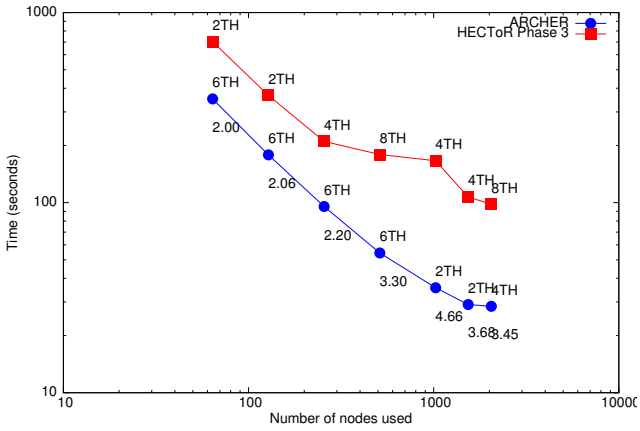

Figure 7. Performance comparison for the H2O-DFT-LS benchmark.

Again examining the results we find that ARCHER is always at least two times faster than HECTOR Phase 3 and scales better at larger (>128 nodes) processor counts due to the improved performance of the Aries/Dragonfly interconnect over the Gemini torus on HECToR Phase 3. In particular, HECToR Phase 3 gave very poor performance between 256 and 1024 nodes due to increased MPI communication costs. As a result, on 1024 nodes ARCHER was 4.66 times faster than HECToR Phase 3.

E. H2O-64-RI-MP2

The H2O-64-RI-MP2 benchmark is a single-point energy calculation using 2nd order Møller-Plesset perturbation theory (MP2) with the Resolution-of-the-Identity approximation to calculate the exchange-correlation energy. The system consists of 64 water molecules in a 12.4 Å^3 cell. This is exactly the same system as used by H2O-64 but using a much more accurate model, which is around 100 times more computationally demanding than standard DFT calculations. The performance of H2O-64-RI-MP2 (see Figure 8) again shows ARCHER being around twice as fast as HECToR Phase 3, although not scaling as well. When using more than 128 nodes using threads was always found to be beneficial to performance.

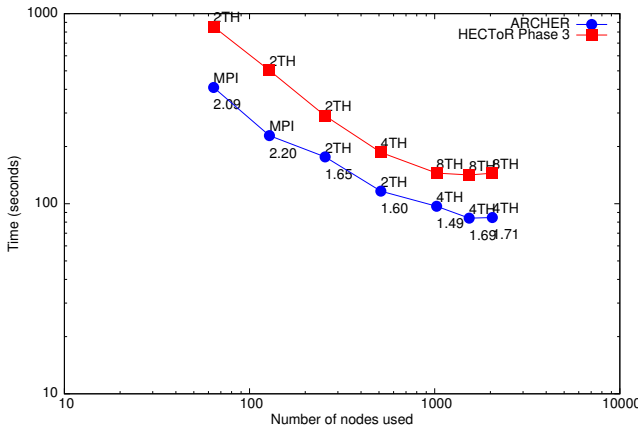

Figure 8. Performance comparison for the H2O-64-RI-MP2 benchmark.

Overall, our benchmarking comparison found ARCHER to be 2-3 times faster in terms of CP2K runtime than HECToR Phase 3. This is largely expected as the peak performance of ARCHER is rougly twice that of HECToR Phase 3 (see Table I). The increased memory per node on ARCHER relative to HECToR Phase 3 allows problem sizes that were previous infeasible to be run. ARCHER was generally found to scale better and to larger processor counts than HECToR Phase 3. This is due to the improved performance of the Aries/Dragonfly interconnect compared with the Gemini torus. For problems using larger numbers of MPI processes (>1000) where the limit of MPI scaling may be reached, the mixed-mode MPI/OpenMP version of CP2K will generally give improved performance over the pure MPI version. Indeed, for problems that are particularly memory intensive (e.g. LiH-HFX) the threaded version may be the only way to successfully run a calculation.

VI. CP2K with Accelerators

Heterogeneous systems are now well established in the HPC community as a concrete alternative to homogeneous systems to increase compute capability with better energy-efficiency. The CSCS Cray XC30 system Piz Daint was built and configured in the fall of 2013 in time for the TOP500 and the Green 500 lists that were published in November

2013. The system is the only machine that appears in the top 10 of both of these lists – where it is the sixth most powerful and the fourth most energy-efficient, respectively. Each node of the system is configured with a single-socket Sandy-Bridge CPU and an NVIDIA K20X GPU (see table I). Therefore the GPU replaces the other CPU socket in the node when compared to a homogeneous XC30 system such as ARCHER. The same approach is used for the XK7 system Tödi at CSCS.

CP2K was one of the applications used during the initial validation phase of Piz Daint. In particular benchmarks based on linear-scaling DFT were chosen because of their potential to scale to very large number of atoms, and suitability for intensive calculations on the GPU. These benchmarks make use of the DBCSR library, which was ported to GPU as consequence of this activity using NVIDIA's CUDA programming model. A detailed description of the implementation and an analysis of the various computational and hardware aspects that influence performance are presented in [15]. Here, we provide a short high-level description of the implementation which is included in the CP2K SVN trunk and freely available from the CP2K website. Recalling what is already described in section II-A about the DBCSR library, the local sparse matrix-matrix multiplication is performed block by block. In the GPU implementation, the CPU is used to traverse the matrix structure, and generate stacks of small matrix multiplications to be performed by the GPU. Data is organized in such a way that the transfers between the host and the GPU are minimized. A double-buffering technique, based CUDA streams and events, is used to maximize the occupancy of the GPU and hide the data transfer latency. Similar to the `libsmm` which is used by the CPU implementation, a corresponding auto-tuned library of small matrix multiplication kernels has been developed for the GPU (`libcusmm`). The overall performance of the library depends strongly on block size, usually 2x–4x faster than batched DGEMM using NVIDIA's cuBLAS library. When the GPU is fully loaded computation may be done simultaneously on the CPU by using the standard algorithm with `libsmm`. Each entire local multiplication is overlapped with the communication step in Cannon's algorithm, therefore it becomes crucial to have a fast enough network data transfer that this can be entirely hidden by the local computation step.

The results for the tests we present here are based on the mixed-mode MPI/OpenMP version of CP2K. We have used both of the two CSCS systems equipped with GPUs: Tödi and Piz Daint. Since we are also interested in a comparison between homogeneous (CPU only) and heterogeneous (CPU+GPU) systems, the tests are executed in two configurations:

- *CPU only (GPU idle)*. The standard CPU code runs on the single-socket CPU per node with 2 OpenMP threads and 4 (Piz Daint) or 8 (Tödi) MPI ranks per node. The GPUs are not involved in the evaluations and they remain idle.
- *CPU+GPU*. Execution of the CUDA-optimized version for simultaneous evaluations on the CPUs and GPUs, with 8 OpenMP threads and 1 (Piz Daint) or 2 (Tödi) MPI ranks per node.

In both configurations the corresponding MPI rank distributions are chosen to reach best performance over the number of nodes used in the tests.

In addition to the `H2O-DFT-LS` already described in the previous section, we use the `AMORPH` (a somewhat larger system with 13846 atoms) and `TiO2` (9786 atoms) benchmarks, both of which use a DZVP-MOLOPT basis set. These are all available as part of the CP2K source distribution in `cp2k/tests/QS/benchmark_DM_LS`. The three benchmarks result in sparse matrices with different block sizes, which impact differently on the overall performance and scalability of the *CPU+GPU* executions. Small blocks do not perform as well on the GPU, while benchmarks with large blocks show worse scalability because of the higher demand on the network communications. Table II reports the block sizes and some performance considerations for each benchmark.

Table II
DIMENSIONS OF THE SMALL MATRIX MULTIPLICATIONS (m, n, AND k VALUES) FOR EACH BENCHMARK.

Benchmark	m, n, k	Note
AMORPH	$m = \{5, 13\}$ $n = \{5, 13\}$ $k = \{5, 13\}$	Small blocks
H2O-DFT-LS	$m = n = k = 23$	Communication limited
TiO2	$m = \{13, 26\}$ $n = \{13, 26\}$ $k = \{13, 26\}$	Balanced

Figures 9 and 10 show the runtime of the benchmarks for the *CPU only* and *CPU+GPU* configurations on Piz Daint and Tödi, respectively. The ratios between the performance of the two configurations for each benchmark and system are shown in Figure 11. The configurations are compared by considering the runtime with $2N$ nodes for the *CPU only (GPU idle)* configuration with respect to N nodes for the *CPU+GPU* configuration. In this way we keep the total number of sockets used the same in each configuration, which gives a fair comparison of whether using a GPU in the second socket on the node is beneficial or not. As expected, the speedup from using a GPU decreases on larger number of nodes as communication becomes dominant and less workload is available for the GPUs. This is particularly evident for `H2O-DFT-LS`. The `AMORPH` benchmark suffers because the small block sizes reduce the efficiency of the CUDA code, while the `TiO2` benchmark has good performance for

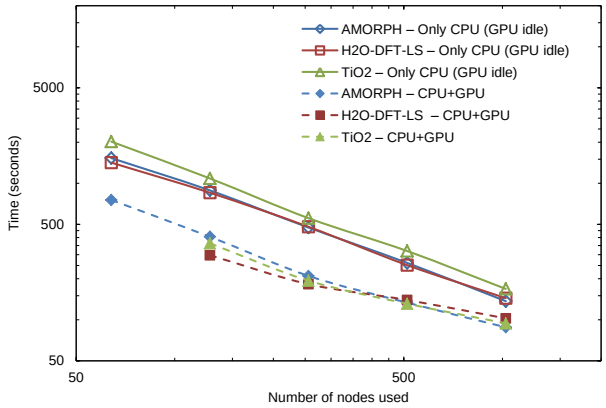

Figure 9. Performance of the benchmarks on Piz Daint, from 64 up to 1024 nodes. H2O-DFT-LS and TiO2 cannot be run on 64 nodes because of the limited GPU memory (6 GB) available.

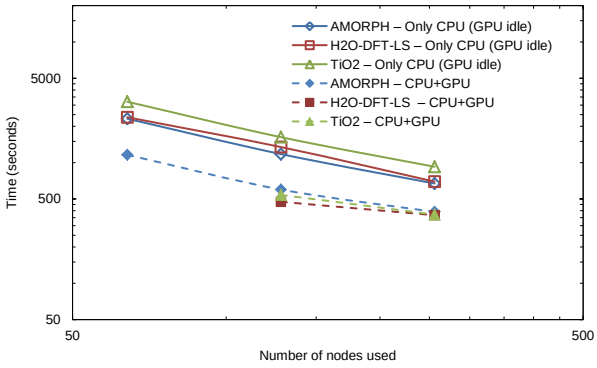

Figure 10. Performance of the benchmarks on Tödi, from 64 up to 256 nodes. H2O-DFT-LS and TiO2 cannot be run on 64 nodes because of the limited GPU memory (6 GB) available.

the *CPU+GPU* configuration, reaching a speed-up of 1.65 on 256 nodes of Piz Daint, and 1.71 on 128 nodes of Tödi.

Figure 12 shows the comparison of the performance between the two systems for the *CPU+GPU* configuration. Execution is around 50% faster on Piz Daint than on Tödi for 64 nodes, increasing to twice as fast for 256 nodes. This is purely down to the faster CPU and network communications available on Piz Daint, since both systems have identical GPUs.

We can also analyze the performance of the DBCSR library specifically. Table III reports the total number of small matrix multiplications, the FLOPs executed in DBCSR part only, and sustained performance of the DBCSR part with respect to the total runtime for each benchmark as measured with the *CPU+GPU* configuration on 1024 nodes of Piz Daint. Table IV reports the fractions of FLOPs executed on the GPUs for the Piz Daint benchmarks. We can see that this increases with the number of nodes involved

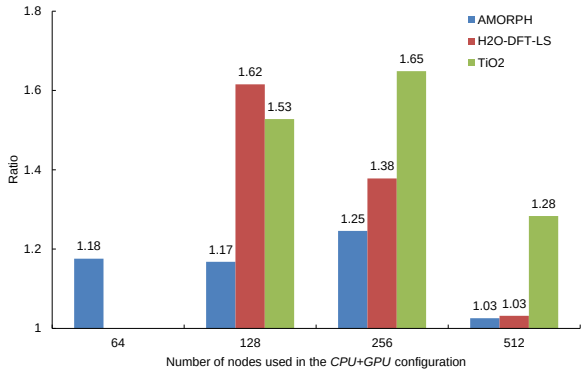

Figure 11. Performance comparison of *CPU+GPU* and *CPU Only* configurations on Piz Daint. The corresponding values on 128 nodes of Tödi are: 1.12 for AMORPH, 1.47 for H2O-DFT-LS, 1.71 for TiO2. The ratio is the time taken on $2N$ nodes for the *CPU only (GPU idle)* configuration divided by the time on N nodes for the *CPU+GPU* configuration. H2O-DFT-LS and TiO2 are omitted on 64 nodes due to the GPU memory limit (6 GB).

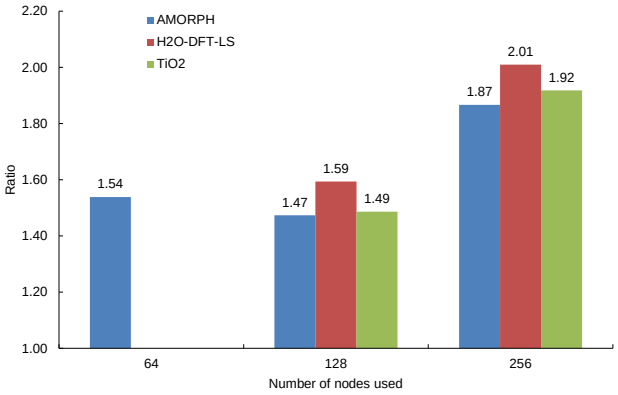

Figure 12. Ratios of the performance of Tödi and Piz Daint for the *CPU+GPU* configuration. GPU memory limit (6 GB) is triggered in the H2O-DFT-LS and TiO2 tests for 64 nodes.

in the tests, and at 1024 nodes almost all small matrix multiplications are executed on the GPUs. On Tödi this is the case already with 256 nodes on Tödi due to the poorer CPU performance of that system.

Table III
PERFORMANCE OF THE DBCSR LIBRARY FOR THE *CPU+GPU* CONFIGURATION ON 1024 NODES OF PIZ DAINT.

	AMORPH	H2O-DFT-LS	TiO2
Number of SMM	1.62×10^{12}	1.44×10^{11}	5.78×10^{11}
DBCSR PFLOP	3.20	3.50	7.24
Sustained TFLOP/s	36.26	34.15	76.82

VII. SUMMARY AND OUTLOOK

In conclusion, we have presented an overview of the main aspects of CP2K which have incrementally improved

Table IV
FLOPS EXECUTED ON THE GPUS (%) FOR PIZ DAINT TESTS.

Number of nodes	AMORPH	H2O-DFT-LS	TiO2
64	87.0		
128	91.5	99.1	89.7
256	95.5	99.7	93.7
512	99.5	100.0	98.4
1024	99.8	100.0	99.2

the performance and scaling of the code, some of which are algorithmic in nature, and others related to optimised implementation and parallelisation in response to architectural changes such as the increasingly wide multi-core nodes present in modern HPC systems. We also discussed recent efforts to enable CP2K for use with CUDA GPU systems. Work is currently ongoing to perform a similar port to Intel's Xeon Phi architecture, and this is expected to be merged into the SVN trunk soon.

An extensive collection of historical data from ab-initio molecular dynamics calculations on liquid water has been presented, covering a period of 9 years, and 7 different Cray system architectures, from the original XT3 up to the present day XC30. A combination of hardware performance improvement as well as software changes resulted in order-of-magnitude speedups which will benefit CP2K's wide user base on the EPCC and CSCS Cray systems, as well as elsewhere.

We have presented some preliminary results on the XE6 and XC30 from the development of a new comprehensive CP2K benchmark suite. In the future we plan to add results from additional HPC systems and publish these on the CP2K website so that users can make an informed decision about the best choice of hardware, MPI and OpenMP combinations for their own work.

Finally, we have shown that for certain classes of calculations, the CUDA capabilities implemented in CP2K enable effective use of heterogeneous architectures with both CPUs and GPUs, and that the XC30 architecture which couples a single socket Intel CPU with an NVIDIA Kepler GPU can give excellent performance for calculations such as linear scaling DFT, which are dominated by DBCSR sparse matrix operations.

Development of CP2K to adapt to changing HPC architectures continues with close collaboration between end-users, developers, HPC centres and vendors via schemes such as ARCHER Embedded Computational Science and Engineering (eCSE), Intel Parallel Computing Centres (IPCC) and the Swiss Platform for Advanced Scientific Computing (PASC). We believe that we have demonstrated the success of this collaborative approach to development, and it will continue to be key as CP2K moves from Petascale towards the Exascale future.

ACKNOWLEDGMENT

The first two authors are supported by the Engineering and Physical Sciences Research Council 'CP2K-UK' project (grant number EP/K038583/1).

This work made use of the facilities of HECToR, the UK's national high-performance computing service, which is provided by UoE HPCx Ltd at the University of Edinburgh, Cray Inc and NAG Ltd, and funded by the Office of Science and Technology through EPSRC's High End Computing Programme.

This work used the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

We are very grateful to Prof. Jürg Hutter and Prof. Joost VandeVondele for providing some of the historical CP2K benchmark data included in this report, and also for access to HPC systems at CSCS for benchmarking and code development.

REFERENCES

[1] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, "CP2K: Atomistic Simulations of Condensed Matter Systems," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 4, no. 1, pp. 15–25, 2014. [Online]. Available: http://dx.doi.org/10.1002/wcms.1159

[2] The CP2K Developers Group. CP2K: Open Source Molecular Dynamics. [Online]. Available: http://www.cp2k.org

[3] M. Bonomi, D. Branduardi, G. Bussi, C. Camilloni, D. Provasi, P. Raiteri, D. Donadio, F. Marinelli, F. Pietrucci, R. A. Broglia, and M. Parrinello, "PLUMED: A portable plugin for free-energy calculations with molecular dynamics," *Computer Physics Communications*, vol. 180, no. 10, pp. 1961 – 1972, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2009.05.011

[4] M. Misic, I. Bethune, and M. Tomasevic, "Automated regression testing and code coverage analysis of the CP2K application," in *Proceedings of the International Conference on Software Testing (to appear)*, 2014.

[5] J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, and J. Hutter, "Quickstep: Fast and accurate density functional calculations using a mixed Gaussian and plane waves approach," *Computer Physics Communications*, vol. 167, no. 2, pp. 103 – 128, 2005. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2004.12.014

[6] J. VandeVondele and J. Hutter, "An efficient orbital transformation method for electronic structure calculations," *The Journal of Chemical Physics*, vol. 118, no. 10, pp. 4365–4369, 2003. [Online]. Available: http://dx.doi.org/10.1063/1.1543154

[7] B. Slater and M. Watkins, "A Quickstep forward: development of the CP2K/Quickstep code and application to ice transport processes (EPSRC grant EP/F011652/1)," 2007-2009, unpublished work.

[8] I. Bethune, "Improving the performance of CP2K on HECToR," Tech. Rep., 2009. [Online]. Available: http://www.hector.ac.uk/cse/distributedcse/reports/cp2k/cp2k_final_report.pdf

[9] N. Li and S. Laizet, "2DECOMP&FFT A highly scalable 2D decomposition library and FFT interface," in *Cray User Group 2010 Conference*, 2010.

[10] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[11] I. Bethune, "Improving the performance of CP2K on multi-core systems," Tech. Rep., 2010. [Online]. Available: http://www.hector.ac.uk/cse/distributedcse/reports/cp2k02/cp2k02_final_report.pdf

[12] U. Borstnik, J. VandeVondele, V. Weber, and J. Hutter, "Sparse matrix multiplication: The distributed block-compressed sparse row library," *Parallel Computing*, no. 0, pp. –, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2014.03.012

[13] I. Bethune, "CP2K - sparse linear algebra on 1000s of cores," Tech. Rep., 2012. [Online]. Available: http://www.hector.ac.uk/cse/distributedcse/reports/cp2k03/cp2k03.pdf

[14] L. E. Cannon, Ph.D. dissertation, 1969.

[15] O. Schutt, P. Messmer, J. Hutter, and J. VandeVondele, "GPU Accelerated Sparse Matrix Matrix Multiplication for Linear Scaling Density Functional Theory," *Submitted*, 2014.

[16] C. Vaughan, M. Rajan, R. Barrett, D. Doerfler, and K. T. Pedretti, "Investigating the impact of the cielo cray xe6 architecture on scientific application codes," in *Workshop on Large-Scale Parallel Processing (LSPP), held in conjunction with the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2011. [Online]. Available: http://www.sandia.gov/~ktpedre/copyrighted-papers/lspp2011.pdf

[17] M. Guidon, J. Hutter, and J. VandeVondele, "Robust Periodic Hartree-Fock Exchange for Large-Scale Simulations Using Gaussian Basis Sets," *Journal of Chemical Theory and Computation*, vol. 5, no. 11, pp. 3010–3021, 2009. [Online]. Available: http://dx.doi.org/10.1021/ct900494g

[18] J. VandeVondele, U. Borstnik, and J. Hutter, "Linear scaling self-consistent field calculations for millions of atoms in the condensed phase," *The Journal of Chemical Theory and Computation*, vol. 8, no. 10, pp. 3565–3573, 2012. [Online]. Available: http://dx.doi.org/10.1021/ct200897x

[19] M. Del Ben, J. Hutter, and J. VandeVondele, "Second-Order Møller-Plesset Perturbation Theory in the Condensed Phase: An Efficient and Massively Parallel Gaussian and Plane Waves Approach," *Journal of Chemical Theory and Computation*, vol. 8, no. 11, pp. 4177–4188, 2012. [Online]. Available: http://dx.doi.org/10.1021/ct300531w

[20] M. Guidon, J. Hutter, and J. VandeVondele, "Auxiliary Density Matrix Methods for HartreeFock Exchange Calculations," *Journal of Chemical Theory and Computation*, vol. 6, no. 8, pp. 2348–2364, 2010. [Online]. Available: http://dx.doi.org/10.1021/ct1002225