First Experiences with *ab initio* Molecular Dynamics on OpenPOWER: The Case of CPMD

Valéry Weber[1], A. Cristiano I. Malossi[1(✉)], Ivano Tavernelli[1], Teodoro Laino[1], Costas Bekas[1], Manish Modani[2], Nina Wilner[3], Tom Heller[3], and Alessandro Curioni[1]

[1] IBM Research–Zurich, Zurich, Switzerland
{vwe,acm,ita,teo,bek,cur}@zurich.ibm.com
[2] IBM India, Bengaluru, India
mamodani@in.ibm.com
[3] IBM, Armonk, USA
{new_nina,tjheller}@us.ibm.com

Abstract. In this article, we present the algorithmic adaptation and code re-engineering required for porting highly successful and popular planewave codes to next-generation heterogeneous OpenPOWER architectures that foster acceleration and high bandwidth links to GPUs. Here we focus on CPMD as the most representative software for *ab initio* molecular dynamics simulations. We have ported the construction of the electronic density, the application of the potential to the wavefunctions and the orthogonalization procedure to the GPU. The different GPU kernels consist mainly of fast Fourier transforms (FFT) and basic linear algebra operations (BLAS). The performance of the new implementation obtained on Firestone (POWER8/Tesla) is discussed. We show that the communication between the host and the GPU contributes a large fraction of the total run time. We expect a strong attenuation of the communication bottleneck when the NVLink high-speed interconnect will be available.

Keywords: CPMD · POWER8 · CUDA · NVlink · FFT · Gram–Schmidt

1 Introduction

Ab initio molecular dynamics (AIMD) is still one of the most commonly used approaches for calculating the time evolution of molecular and solid state systems under ambient conditions of temperature and pressure. Specifically, AIMD is particularly suited for the simulation of complex molecular systems that undergo

M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS 9945, pp. 228–234, 2016.
DOI: 10.1007/978-3-319-46079-6_16

important reorganizations of their electronic structure (bond breaking and bond forming), and for which the design of a classical force field is very cumbersome. Essential to all AIMD techniques is the calculation of the molecular potential and the corresponding forces obtained from the derivatives of the potential with respect to the nuclear coordinates. These forces are then used to solve the Newton equation of motion for calculating of the nuclear trajectories.

Within density functional theory (DFT), the molecular Hamiltonian is mapped (in principle exactly) into a system of noninteracting particles subject to a compensating local external potential (the exchange-correlation (xc) potential), for which we need approximations.

CPMD [1] uses a pseudopotential-based Kohn–Sham DFT description of the electronic structure in which the Kohn–Sham orbital and the electronic density are expanded in a plane-wave basis set. In addition, working with plane waves has the important advantage of simplifying the calculation of energies and forces; thus some parts of the total energy (such as the kinetic term) are efficiently computed in the Fourier (reciprocal) space, whereas other parts, like the Hartree energy and the interaction with external fields, are accurately evaluated in the real (direct) space. The limiting steps in the plane-wave implementation of AIMD codes consist in (a) the forward and backward Fourier transforms (FFT) [3] (wavefunctions, potentials, and energy terms) and (b) the orthogonalization of the wavefunctions.

The combined use of plane waves and pseudopotential together with highly optimized algorithms for the computation of energies and forces made CPMD one of the most efficient DFT-based AIMD codes, with a documented scaling performance that extends to one million computing cores [5].

The advent of data-centric OpenPOWER systems based on the IBM, NVIDIA and MELLANOX collaboration offers a new potential for scalability and performance that leads to Exascale systems. Here, we present our strategy plans in migrating CPMD to the data-centric systems and summarize our progress so far.

2 Methodology

The basic task in Kohn–Sham based DFT is the minimization of the energy density functional with respect to the Kohn–Sham orbitals $\{\phi_i(\mathbf{r})\}$

$$E_{\text{tot}} = \min_{\{\phi_i\}} E^{KS}\left[\{\phi_i(\mathbf{r})\}\right], \tag{1}$$

where

$$E[\{\phi_i(\mathbf{r})\}] = T_s[\{\phi_i(\mathbf{r})\}] + \int d\mathbf{r}\, V_{ext}[\{\phi_i(\mathbf{r})\}]\rho(\mathbf{r}) + \int d\mathbf{r}\, V_H[\{\phi_i(\mathbf{r})\}]\rho(\mathbf{r}) + E_{\text{XC}}[\{\phi_i(\mathbf{r})\}],$$

and $T_s[\{\phi_i(\mathbf{r})\}]$ is the kinetic energy, $V_{ext}[\{\phi_i(\mathbf{r})\}]$ is the external potential generated by the nuclei, $V_H[\{\phi_i(\mathbf{r})\}]$ is the Hartree (Coulomb) term, $E_{\text{XC}}[\{\phi_i(\mathbf{r})\}]$ is

the exchange correlation energy, and $\rho(\mathbf{r}) = \sum_{i=1}^{N} |\phi_i(\mathbf{r})|^2$ is the electronic density. Index i runs through the N states of the system. The minimization of (1) leads to a self-consistent set of Kohn–Sham equations

$$\begin{aligned} \left[-\frac{1}{2}\nabla_i^2 + V_{\text{eff}}[\rho]\right] \phi_i^{KS}(\mathbf{r}) &= \epsilon_i \phi_i(\mathbf{r})^{KS}, \\ V_{\text{eff}}[\rho] &= V_{ext} + V_H[\rho] + V_{xc}[\rho], \\ V_{xc}[\rho] &= \frac{\delta E_{xc}[\rho(\mathbf{r})]}{\delta \rho(\mathbf{r})}. \end{aligned} \tag{2}$$

Numerically, the solution of the Kohn–Sham equations (2) requires a direct minimization algorithm that preserves the orthogonality among the Kohn–Sham orbitals. To optimize this process, Bekas and Curioni [2] recently proposed a block variant of Gram–Schmidt that ensures high processor performance and excellent scaling. The new method exploits data locality to allow the best mapping on the cache-memory hierarchies of modern processors and also enable optimal utilization of the memory subsystem of hardware accelerators such as GPUs. Unlike the current state of the art, the simplicity of the new schemes, inherited from the original Gram–Schmidt method, renders them ideal for enabling much needed fault-tolerance properties when they are deployed on massively parallel computing systems.

High efficiency and numerical scalability in CPMD are achieved thanks to the use of the plane-wave basis set for the expansion of the Kohn–Sham orbitals

$$\phi_i(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} \sum_{\mathbf{G}} \tilde{\phi}_i(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{r}},$$

where Ω is the volume of the simulation box, and $\mathbf{G}$ is the index that runs through the reciprocal space vectors. The Fourier coefficients $\tilde{\phi}_i(\mathbf{G})$ are then related to the Kohn–Sham orbital through the inverse FFT

$$\tilde{\phi}_i(\mathbf{G}) = \frac{1}{\sqrt{\Omega}} \sum_{\mathbf{r}} \phi_i(\mathbf{r}) e^{-i\mathbf{G}\cdot\mathbf{r}}.$$

The number of operations required for the conversion of a general function f

$$f(\mathbf{r}) \underset{\text{invFFT}}{\overset{\text{FFT}}{\Longleftrightarrow}} \tilde{f}(\mathbf{G}),$$

using 3d FFT is approximately on the order of $5N \log N$, where N is the number of grid points in the direct space.

3 GPU Implementation

Achieving overlap between data transfer and computation requires the use of CUDA streams. A stream is a sequence of operations that are executed on the

GPU in the same order as they are launched from the host. Operations between streams can be interleaved and potentially run concurrently. Below we summarize the three main computational kernels that we have ported to CUDA so far. In the following, on-GPU FFT transforms and BLAS operations kernels are implemented in the cuFFT and cuBLAS libraries, respectively.

Construction of the Electronic Density. The reverse Fourier transform of the N states $\phi_i(\mathbf{r})$ is distributed over the N_S streams that work concurrently. Each stream is assigned to a CPU thread and performs the sequence of operations needed to transform a state $\tilde{\phi}_i(\mathbf{G})$ to the corresponding state density $\rho_i(\mathbf{r}) = |\phi_i(\mathbf{r})|^2 = |\text{invFFT}(\tilde{\phi}_i(\mathbf{G}))^2|$. The summation over i of the N state densities finally gives the desired electronic density $\rho(\mathbf{r})$.

The computation of a state density starts with an asynchronous communication of a state $\phi_i(\mathbf{G})$ from the host to the device; the GPU performs an 1-D FFT and then back copies the data to the host. The host proceeds to the interprocess communication (MPI all-to-all) while taking care of packing/unpacking operations. We note that a specific MPI communicator is assigned to each CPU thread. In the last phase, the data is transferred to the GPU, which performs a 2d FFT before pushing the direct-space state to the host. The host finally adds up the N contributions to the electronic density.

Applying the Potential to the Wavefunctions. The reverse and forward Fourier transforms as well as the application of the potential $V_{\text{eff}}[\rho]$ to the N states are distributed over N_S streams that work concurrently. Each stream is again assigned to a CPU thread and performs the sequence of operations needed to apply the potential to a state $V\phi_i(\mathbf{G}) = \text{FFT}(V\text{invFFT}(\tilde{\phi}_i(\mathbf{G})))$. Thus the reverse FFT of a state from reciprocal to direct space is identical to the construction of the electronic density. The direct-space state is then copied to the host, and the potential is applied. The forward transform takes place by performing the 2d FFT (on the GPU), followed by the interprocess communication (on the host) and the last 1d FFT (on the GPU).

Orthogonalization. We modified the block Gram–Schmidt scheme introduced in [2] to make use of the GPU. Let us assume that only one MPI task is used (generalization to multiple MPI tasks is trivial). The coefficient matrix (which corresponds to the coefficients of the expansion of ϕ_i on the plane-wave basis) is block-partitioned column-wise into n blocks of size b as $C = [C_1, C_2, \ldots, C_n]$. We seek the orthogonalized coefficient matrix $\tilde{C} = \text{ortho}(C)$. The rows of the coefficient matrix C are block distributed over CPU threads. Each CPU thread is assigned three streams. The first stream, which we refer to as S_{cmp}, is used for computation, while the other two streams are in charge of host-to-device (S_{h2d}) and device-to-host (S_{d2h}) asynchronous communications of the C and $\tilde{C}$ matrices, respectively. The key idea of the block Gram–Schmidt scheme is to loop over the n blocks C_i and to orthogonalize them one after the other.

The orthogonalization is done by first projecting out the previously orthogonalized blocks $[\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{i-1}]$ and then using a Cholesky (of size $b \times b$) based orthogonalization to produce the $\tilde{C}_i$. In each iteration, intermediate reductions among the threads (the row distribution of the matrix) are needed. The stream S_{cmp} is in charge of performing the BLAS operations as well as the intermediate communication for reductions. The role of the stream S_{h2d} is to asynchronously copy the block C_{i+1} to the GPU for the next orthogonalization iteration. The stream S_{d2h} is used to copy $\tilde{C}_{i-1}$ back to the host.

4 Results

To illustrate the progress on porting CPMD to OpenPOWER systems, we show the strong scaling of the construction of the density, the application of the potential to the wavefunctions and the orthogonalization process for a box of 128 water molecules at normal liquid density and under periodic boundary conditions. We use the GTH pseudopotential [4] and plane-wave and density cutoffs of 100 Ry and 400 Ry, respectively.

The code is compiled using the IBM XL Fortran compiler for Linux 15.1.4 with optimization flags: `-O3 -qhot -qstrict -qprefetch=aggressive:dscr=7 -qsimd=auto -qaltivec -qmaxmem=-1 -qsmp=omp`. The C-code was compiled with the GNU compiler collections 4.9.3. The runs are performed on two IBM POWER8 systems: Tuleta and Firestone. Both servers are equipped with two POWER8 processors. Each POWER8 core supports 8 hardware threads, has 64 kBytes L1 cache, 512 kBytes L2 cache, and 8 MBytes of shared L3 cache. Tuleta runs 12 cores in total at 4.2 GHz, whereas Firestone equips 20 cores at 3.42 GHz. Tuleta has one Nvidia Tesla K40, with 2880 CUDA cores; Firestone has two Nvidia Tesla K80 GPUs, each composed of two devices with 2496 CUDA cores. All computations are performed with CUDA compute capability 3.5 (on Tuleta) and 3.7 (on Firestone), both with driver version 7.5. On Firestone, our calculations use only one device of one K80, i.e., 2496 CUDA cores.

The performance comparison for the three computational blocks described in Sect. 3 is shown in Fig. 1. First, we observe that the Firestone CPU performance is better than that of the earlier Tuleta processor for the two FFT computational blocks. Concerning the GPU results, we observe a dependence on the number of streams used. The PCI-E bandwidth in the two systems is equivalent; therefore once it is saturated, the K40 tends to run slightly faster than half-K80, because of the slightly greater number of CUDA cores. The optimal number of streams varies between 4 and 6, depending on the type of computational block. Using more streams does not help, as memory bandwidth becomes the limiting factor. By analyzing the output of NVprof, we summarize, in Table 1, the time percentage spent in computation and memory copies for the construction of the electronic density and applying the potential to the wavefunctions. Although all operations are performed asynchronously, the time spent in memory copies exceeds the computation time by far (about 1/3 of the total time), so that for at least 2/3 of the total time the GPU cores are idle, waiting for data.

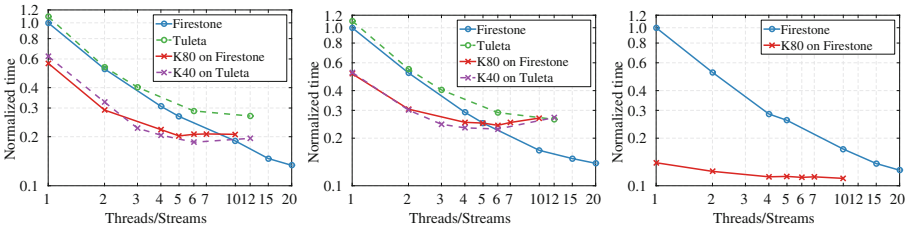

Fig. 1. Log-scale performance comparison of three CPMD computational blocks run on IBM Firestone (20-cores vs. 1-device K80) and IBM Tuleta (12-cores vs. K40). Time is normalized w.r.t. one core CPU time on Firestone. All CPU runs are performed with one thread per core. All GPU runs are performed with one stream per thread per core. (a) Construction of the electronic density. (b) Applying the potential to wavefunctions. (c) Orthogonalization (results on Tuleta are not available)

Table 1. Time percentage spent in computation and memory copy from device to host (D2H) and host to device (H2D) for constructing the electronic density and applying the potential to the wavefunctions. Computation and memory copies are performed asynchronously.

Kernel	Computation [%]	D2H [%]	H2D [%]
Electronic density	27	95	76
Applying potential	30	92	89

At the current state of development, maximum performance is achieved by the 2-socket CPU of Firestone for the FFT computational kernels. We expect that future improvements in the CUDA implementation, including the generalization to multi-GPUs, will change this picture in favor of the accelerators.

5 Future Works

Our initial porting phase of CPMD to OpenPOWER architectures highlights the negative impact of a limited PCI-E bandwidth between the CPU and the GPU. To alleviate this problem, we will tackle the issue from multiple directions: at the implementation level, we will move the calculation of the electronic density and the application of the potential to the wavefunctions to the GPUs and, more generally, we will minimize data transfer whenever possible. At the architecture level, we expect a significant improvement from the NVLink high-speed interconnect equipped by next-generation Garrison POWER8' systems. NVLink will enable ultra-fast communication between the CPU and GPU, allowing data transfer at rates more than 2.5 times faster than traditional PCI-E interconnects. This should be tremendously beneficial for scenarios such as the one summarized in Table 1, where multiple streams have overlapped operations, but communication time is left exposed. Garrison's systems will be available in Q3–Q4 2016:

the study and characterization of the performance gain obtained on such an architecture will be subject of future work.

References

1. CPMD ver. 4.1: Copyright IBM Corp.1990–2016, Copyright MPI für Festkörperforschung Stuttgart (1997–2001). http://www.cpmd.org
2. Bekas, C., Curioni, A.: Very large scale wavefunction orthogonalization in density functional theory electronic structure calculations. Comput. Phys. Commun. **181**(6), 1057–1068 (2010)
3. Goedecker, S.: Fast radix 2, 3, 4, and 5 kernels for fast Fourier transformations on computers with overlapping multiply-add instructions. SIAM J. Sci. Comput. **18**(6), 1605–1611 (1997)
4. Goedecker, S., Teter, M., Hutter, J.: Separable dual-space Gaussian pseudopotentials. Phys. Rev. B **54**, 1703–1710 (1996)
5. Weber, V., Bekas, C., Laino, T., Curioni, A., Bertsch, A., Futral, S.: Shedding light on lithium/air batteries using millions of threads on the BG/Q supercomputer. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 735–744 (2014)