SC16

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

Valéry Weber†, A. Cristiano I. Malossi†, Ivano Tavernelli†, Teodoro Laino†, Costas Bekas†, Manish Modani‡, Nina Wilner*, Tom Heller*, Alessandro Curioni†

† IBM Research – Zurich, ‡ IBM India, Bengaluru, India, * IBM USA

Introduction

The CPMD code is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. The algorithmic adaptation and code re-engineering to port CPMD code to next-generation heterogeneous Open-POWER architectures (CPU+GPU) is presented here. The construction of the electronic density, the application of the potential to the wavefunctions and the orthogonalization procedure are offloaded 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) and Minsky (POWER8+/Pascal) is discussed. As expected, the communication between the host and the GPU contributes to a large fraction of the total computation time for PCI based systems. We observe a strong attenuation of the communication bottleneck when the NVLink high-speed interconnect, between CPU to GPU is beging used. (Some of the results obtained with the Firestone system have been presented at ISC16 workshop.)

Summary

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 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

CPMD [1] uses a pseudopotential-based Kohn–Sham DFT description of the electronic structure in which the KohnâĂŞShamorbital 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 Ab initio molecular dynamics (AIMD) codes consist in

1. the forward and backward Fourier transforms (FFT) [3] (wavefunctions, potentials, and energy terms) and
2. 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 [5].

Methodology and Equations

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

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

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

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

The solution of the Kohn–Sham equations (2) requires a direct minimization algorithm that preserves the orthogonality among the Kohn–Sham orbitals. Bekas and Curioni [2] recently proposed a block variant of the modified Gram–Schmidt that ensures high processor performance and excellent scaling. 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

$$f(\mathbf{r}) \xleftrightarrow[\text{invFFT}]{\text{FFT}} \tilde{f}(\mathbf{G}),$$

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

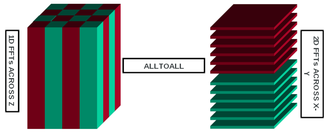

Figure 1: Distribution of the 3D-FFT coefficients on 2 MPI tasks.

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

As described in Fig. 2, 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 $\tilde{\phi}_i(\mathbf{G})$ 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.

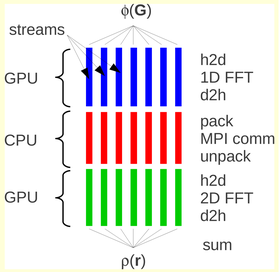

Figure 2: Diagram of the FFT streams pipeline.

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: modified block Gram–Schmidt

We seek the orthogonalized coefficient matrix $\tilde{C} = \text{ortho}(C)$ [2]. The rows of the coefficient matrix C are block distributed over CPU threads/MPI tasks. 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 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/MPI tasks (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.

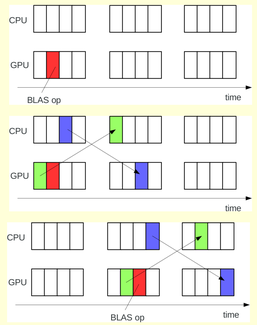

Figure 3: Orthogonalization scheme.

Architecture and Setting

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 runs are performed on two IBM POWER8 systems: Tuleta (one Tesla K40) and Firestone (four logical Tesla K80), and IBM POWER8+ system: Minsky (four Tesla P100).

Results

Single GPU configuration

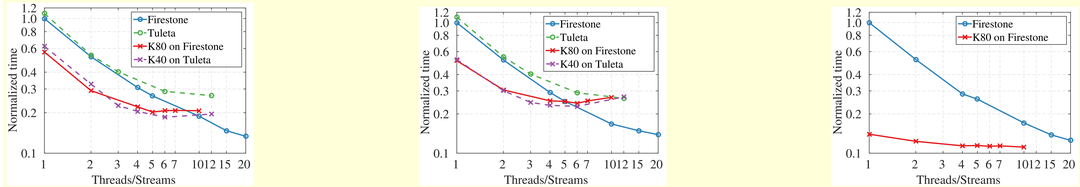

Figure 4: Log-scale performance comparison of three CPMD computational blocks run on IBM Firestone (20-cores vs. 1-logical K80) and IBM Tuleta (12-cores vs. K40). Time is normalized w.r.t. one core CPU time on Firestone. All CPU runs are performed with one thread per core. All GPU runs are performed 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).

The performance comparison for the three computational blocks described before is shown in Fig. 4. 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.

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

Multi GPU configuration

The implementation of CPMD is curently beging generalized for Multi-GPU. Preliminary timing obtained with the Minsky system for multiple GPUs / MPI tasks is shown in Fig 2. The performance is compared with the best performance observed with CPU only version of the code.

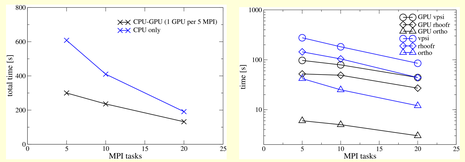

Figure 5: CPMD performance for CPU only version compared with CPU/GPU version with the increase in number of GPUs. Total execution time (left) and timing for different kernels (right). 5 MPI tasks are assigned per GPU. Time measured on IBM Minsky (20–cores with 4-devices P100)

Figure 6: Timeline view of the construction of the density (rhoofr) and application of the potential to the wavefuctions (vpsi) obtained with NVprof.

References

[1] CPMD ver. 4.1, `http://www.cpmd.org`. Copyright IBM Corp. 1990–2016, Copyright MPI für Festkörperforschung Stuttgart 1997–2001.

[2] C. Bekas and A. Curioni. Very large scale wavefunction orthogonalization in density functional theory electronic structure calculations. *Computer Physics Communications*, 181(6):1057–1068, 2010.

[3] S. Goedecker. Fast radix 2, 3, 4, and 5 kernels for fast Fourier transformations on computers with overlapping multiply–add instructions. *SIAM Journal on Scientific Computing*, 18(6):1605–1611, 1997.

[4] S. Goedecker, M. Teter, and J. Hutter. Separable dual-space Gaussian pseudopotentials. *Physical Review B*, 54:1703–1710, 1996.

[5] V. Weber, C. Bekas, T. Laino, A. Curioni, A. Bertsch, and S. Futral. Shedding light on lithium/air batteries using millions of threads on the BG/Q supercomputer. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 735–744, 2014.