Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–18

DOI: 10.1177/1094342016649420
hpc.sagepub.com
SAGE

Efficient parallelization of perturbative Monte Carlo QM/MM simulations in heterogeneous platforms

Sebastião Miranda[1], Jonas Feldt[2], Frederico Pratas[1], Ricardo A. Mata[2], Nuno Roma[1] and Pedro Tomás[1]

Abstract

A novel perturbative Monte Carlo mixed quantum mechanics (QM)/molecular mechanics (MM) approach has been recently developed to simulate molecular systems in complex environments. However, the required accuracy to efficiently simulate such complex molecular systems is usually granted at the cost of long executing times. To alleviate this problem, a new parallelization strategy of multi-level Monte Carlo molecular simulations is herein proposed for heterogeneous systems. It simultaneously exploits fine-grained (at the data level), coarse-grained (at the Markov chain level) and task-grained (pure QM, pure MM and QM/MM procedures) parallelism to ensure an efficient execution in heterogeneous systems composed of central processing units and multiple and possibly different graphical processing units. This is achieved by making use of the OpenCL library, together with appropriate dynamic load balancing schemes. From the conducted evaluation with real benchmarking data, a speed-up of 56x in the computational bottleneck part was observed, which results in a global speed-up of 38x for the whole simulation, reducing the time of a typical simulation from 80 hours to only 2 hours.

Keywords

Quantum mechanics, molecular mechanics, Monte Carlo simulations, parallel computing, openCL, GPGPU, heterogeneous systems

1 Introduction

Computer simulations have become standard tools in chemical research, allowing for the prediction and characterization of complex molecular structures. Advances in this field are not only due to improvements of physical models, but also to improvements in computing systems, which substantially reduce the computational time. Today, thousands of compounds can be screened in a matter of minutes for compatibility and possible activity. This plays a crucial role in drug design (Geromichalos, 2007), to which thousands of lives are tied.

Molecular simulations are commonly based on molecular dynamics (MD) or on the Monte Carlo (MC) method. MD simulates the system by calculating the forces acting on each atom, applying classical mechanics to evolve the system in time. MD allows the study of a wide range of dynamical properties, such as the conformational landscape of a molecule. However, usable results can only be obtained by using very small time steps (in the order of a femtosecond), which limits the system simulation to the order of microseconds. Conversely, the Metropolis MC method (Metropolis et al., 1953) samples the system in the ensemble space, rather than following a time coordinate. With this method, a sequence of random configurations is obtained on the basis of Maxwell-Boltzmann statistics, by performing random movements at each frame and by evaluating the corresponding change of the system energy. The resulting set is then analyzed from the perspective of the specific thermodynamic property under consideration. Even though MC does not enable the computation of dynamical quantities, it allows the

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2]Institut für Physikalische Chemie, Universität Göttingen, Göttingen, Germany

Corresponding author:
Pedro Tomás, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Rua Alves Redol, 9, 1000-029 Lisbon, Portugal.
Email: pedro.tomas@inesc-id.pt

study of processes with longer timescales, for which sampling in time would be infeasible.

Accordingly, the underlying method for calculating the energy of a given molecular structure can vary depending on the system and the properties under study. The choice can fall to traditional molecular mechanics (MM), quantum mechanics (QM) or mixed QM/MM methods. MM approaches represent atoms and molecules through ball and spring models, with heavily parameterized functions to describe their interactions. However, such an approach may lead to several limitations. For example, atomic bonds have to be kept throughout each simulation, thus preventing the chemical reaction being modeled in a single run. Alternatively, QM approaches explicitly simulate the electrons, at a cost of a much higher computational burden, as they involve obtaining approximate solutions to the Schrödinger equation. Furthermore, the computation cost of most QM methods scales exponentially with the system size, thus impeding the modeling of more complex structures. An alternative solution consists of a mixed QM/MM approach, which combines the strengths of each method. In this case, a small active region is simulated with QM, while the remaining environment is represented by classical MM. Nevertheless, the combination of the mixed QM/MM terms with the pure QM and MM terms that co-exist in this approach usually result in a very computationally diverse algorithm containing both heavy single-threaded code and several opportunities to exploit task and data parallelism.

In this context, the perturbative Monte Carlo (PMC) mixed QM/MM method, first suggested by Truong and Stefanovich (1996), was presented as a promising approach to mixed QM/MM calculations. In this method, the coupling between QM and MM is approximated by using perturbation theory, leading to 2-body interaction energy terms. However, the algorithm has to be designed to take full advantage of this separation, and no efficient implementation was ever put forward.

In this work, a multi-device parallel approach to PMC is proposed, by using Metropolis MC sampling and a mixed QM/MM method for the energy calculation. In the presented solution, the QM/MM part of the PMC algorithm (henceforth referred to as the PMC cycle) is accelerated using OpenCL. The QM part (henceforth referred to as the QM update) invokes the widely used MOLPRO (Werner et al., 2012) program package, and it is accelerated by exploiting multiple central processing unit (CPU) cores to run several independent state-space Markov chains. The proposed solution is both versatile and scalable, enabling researchers to combine different generations of computing platforms, accompanying the growing tendency of heterogeneous computational clusters. Accordingly, the main contributions of this work are the following.

1. First parallel heterogeneous solution for the perturbative Monte Carlo QM/MM method.
2. Acceleration procedure based on the simultaneous exploitation of fine-grained (at the data level), course-grained (at the Markov chain level) and task-grained (pure QM, pure MM and QM/MM procedures) parallelism to achieve a scalable solution for heterogeneous platforms.
3. Integration of a performance-aware scheduling algorithm in the parallel PMC simulation to enable the full and balanced exploitation of the computing power of all the different devices in a given heterogeneous platform.
4. Study and evaluation of other acceleration and energy saving opportunities based on the adaptation of the numerical precision used by the algorithm when considering either double or single-precision floating-point or fixed-point representations. Such a study was integrated in the conducted performance and power analysis.

Hence, by properly parameterizing the kernels, the same OpenCL solution may be efficiently run in very different platforms (e.g. CPUs, graphical processing units (GPUs), CPUs + GPUs, etc.), where each of these platforms may constitute an individual processing node of a scalable multi-node processing infrastructure, e.g. CPU/GPU computational cluster managed either using MPI or OpenCL (Kim et al., 2012). In particular, the considered proof-of-principle implementation that is herein presented is focused on tackling the existing parallelization challenges and on demonstrating and evaluating the performance advantages of the proposed CPU + GPU parallelization. In accordance with this, it considers a single processing node equipped with a (AVX2 enabled) quad-core Intel i7-4770K CPU and several different GPU accelerators, from NVIDIA and AMD.

By considering the execution on a single CPU core as a steady baseline reference, it is shown that a speed-up of about 38× is obtained when running multiple Markov chains in parallel, by exploiting the multi-core CPUs to simultaneously run several QM updates, while the GPU accelerators execute independent instances of the PMC cycle. For the longest QM/MM simulation herein discussed, this effectively reduced the full execution time from ~80 hours to ~2 hours.

The rest of this paper is organized as follows: Section 2 provides a brief background on the PMC QM/MM algorithm. Section 3 discusses the existing parallelization opportunities and presents the proposed parallel solution. Furthermore, a dynamic load balancing for multiple heterogeneous accelerators is presented, as well as an approach to sample the MC state-space in parallel using multiple CPU cores. Section 4 describes the benchmarking setup and presents the timing and the

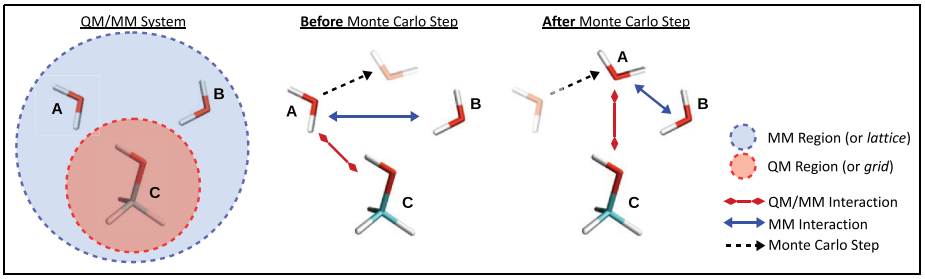


Figure 1. A system composed of one solute molecule (C) and two solvent molecules (A and B). For each MC step, the difference in terms of the energy between the displaced molecule (A) and every other molecule has to be computed, but at different levels of theory.

speed-up results for the simulation bottleneck (PMC cycle) and the complete PMC QM/MM run, when running single and multiple Markov chains. In Section 5, a study about the impact of using mixed precision (single and double precision floating-point, as well as fixed-point) on the execution time and power consumption of the bottleneck procedure is presented. Finally, related work is discussed in Section 6.

2 Perturbative Monte Carlo QM/MM

The present paper is focused on the computational aspects of the PMC algorithm, with a particular emphasis on the acceleration and use of hybrid computing nodes. Nevertheless, a brief characterization of the QM/MM simulations under study, together with an overview of the PMC method, is herein presented.

The perturbative Monte Carlo QM/MM algorithm is a molecular simulation procedure designed to study mixed QM/MM simulations. These interactions usually consider a circumscribed region of interest and an immersive environment. For the purpose of describing the PMC method, a chemical solution composed of a solute (region of interest) and a solvent (environment) will be herein considered as an example. Accordingly, Figure 1 depicts such a system, comprised of a single solute molecule (molecule C), which is treated at the QM level, and two solvent molecules (molecules A and B), treated at the MM level. Then, by applying a Metropolis MC step, one of these molecules will be randomly picked, translated and rotated to generate a new structure. This last MC step will be either accepted or rejected, according to the resulting energy change. This energy change is computed by considering two types of interactions with the displaced molecule (e.g. molecule A), resulting in both QM/MM energy terms or pure MM terms. The QM/MM terms account for the interaction with the QM solute (molecule C), whereas the MM energy terms account for the interaction with every other solvent molecule (in this case, just molecule B). Furthermore, for both levels of theory (QM/MM or pure MM), Coulomb and van der Waals (vdW) contributions have to be considered.

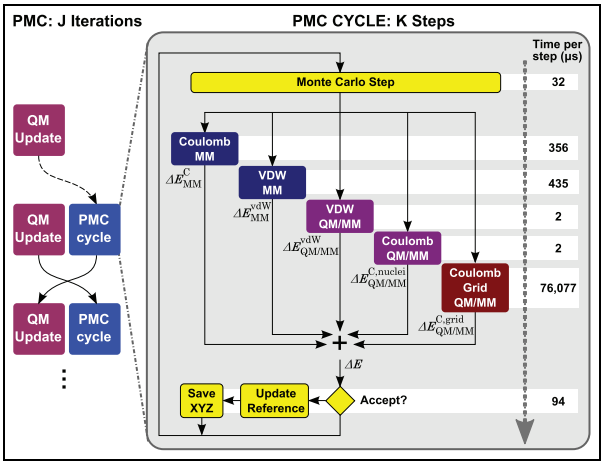


Figure 2. Perturbative Monte Carlo QM/MM method. Time footprint for a single PMC cycle step when processing *bench-A* dataset (see Section 4) running on one core of the *Intel i7-4770K* CPU (AVX2 enabled).

Figure 2 illustrates the dataflow corresponding to the considered PMC method. In each iteration, the PMC cycle comprehends K Monte Carlo steps of the MM subsystem, while keeping the QM region static. In another process, the electronic density of the QM region is updated (*QM update*) by using MOLPRO, and the result is subsequently used in the next PMC cycle. As described earlier, the system energy variation has to be computed at each MC step (henceforth referred to as a PMC cycle step), given by

$$\Delta E = \Delta E^{\mathrm{C}}_{\mathrm{MM}} + \Delta E^{\mathrm{vdW}}_{\mathrm{MM}} + \Delta E^{\mathrm{vdW}}_{\mathrm{QM/MM}} + \Delta E^{\mathrm{C,\,nuclei}}_{\mathrm{QM/MM}} + \Delta E^{\mathrm{C,\,grid}}_{\mathrm{QM/MM}} \quad (1)$$

where each ΔE term corresponds to a program procedure in Figure 2. The C superscript in Eq. (1) refers to Coulomb interaction terms. The most computationally intensive step corresponds to the $\Delta E^{\mathrm{C,\,grid}}_{\mathrm{QM/MM}}$ term calculation. In the approach that is herein presented, the $\Delta E^{\mathrm{C,\,grid}}_{\mathrm{QM/MM}}$ term is computed by expressing the electronic density of the QM system in an atom-centered

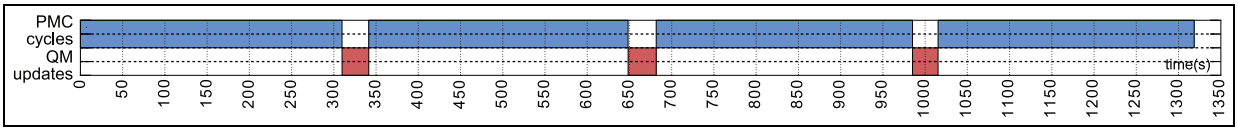


Figure 3. Four complete PMC iterations, each one comprised of a QM update and 4k PMC cycles (see Figure 2) for *bench-A* dataset (see Section 4), running on one single core of the *Intel i7-4770K* CPU (AVX2 enabled). The bottleneck of each PMC iteration is the PMC cycle.

discrete grid, which is subsequently used for computing the interactions with the MM charges.

Figure 2 also includes a profiling evaluation executed on a single-core of an *Intel i7-4770K* CPU with vector instructions enabled (AVX2), when processing the *bench-A* input dataset (see Section 4 for additional details). Figure 3 complements this information by presenting the overall execution results for a few PMC iterations, whereas Figure 2 depicts a more detailed overview of each step of the simulation bottleneck. The Coulomb grid QM/MM procedure ($\Delta E^{\mathrm{C,grid}}_{\mathrm{QM/MM}}$) represents, for all the tested input QM/MM systems, the most time-consuming part of each PMC cycle step. The pseudo code corresponding to this energy computation is presented in Algorithm 1. For each {*atom, grid point*} pair (considering the atoms of the displaced molecule), the Coulomb potential is computed. Furthermore, since periodic QM/MM systems are herein considered (defined by a repeating simulation box), the spacial range of the considered electrostatic interactions (i.e. Coulomb, vdW) has to be limited by a cutoff distance in space (r_c). Accordingly, shifted potentials (V_{shift}) (Fennell and Gezelter, 2006) are used in the $\Delta E^{\mathrm{C,grid}}_{\mathrm{QM/MM}}$ interaction terms

$$V_{\mathrm{shift}} = \begin{cases} \frac{1}{r} - \frac{1}{r_c} + \frac{1}{r_c^2}(r - r_c) & r < r_c \\ 0 & r \geq r_c \end{cases} \qquad (2)$$

affecting each term differently, depending on the distance between each {*atom, grid point*} pair (r), and completely disregarding the interaction (set to 0) whenever $r \geq r_c$. The use of shifted potentials can be observed in Algorithm 1, resulting in four separate space regions depending on the distance between the considered *grid point* and both the old and the new set of coordinates of each *atom* of the displaced molecule. Hence, four slightly different energy expressions (resulting from the application of V_{shift}) may be computed. The $\Delta E^{\mathrm{C}}_{\mathrm{MM}}$ and $\Delta E^{\mathrm{vdW}}_{\mathrm{MM}}$ terms have similar algorithm structures, although the involved data and the corresponding expression for the energy computation vary slightly.

Algorithm 1 Coulomb QM/MM energy contribution.

```
Init: Energy = 0.0
Init: r_c → Coulomb cutoff (run parameter)
  for each atom in changed molecule do
    for each point in charge grid do
      r_old = distance(atom, point) in reference system
      r_new = distance(atom, point) in new system
      qs = − point_charge × atom_charge
      if r_new < r_c and r_old < r_c then
        Energy + = qs × (1/r_new − 1/r_old + 1/r_c^2 (r_new − r_old))
        else if r_new < r_c and r_old ≥ r_c then
          Energy + = qs × (1/r_new − 1/r_c + 1/r_c^2 (r_new − r_c))
        else if r_old < r_c then
          Energy − = qs × (1/r_old − 1/r_c + 1/r_c^2 (r_old − r_c))
    end if
  end for
end for
```

3 PMC cycle parallelization

This section describes the proposed parallelization of the PMC QM/MM algorithm, by exploiting heterogeneous platforms composed of a multi-core CPU and one or more accelerators (e.g. GPUs). The multi-core CPU is responsible for coordinating all the accelerators and for running the QM update procedure, whereas the accelerators run the PMC cycle, which represents the bottleneck of the PMC iteration. Furthermore, in order to ensure a full compatibility with a wide range of different accelerators, the proposed parallel solution for the PMC cycle was developed using the OpenCL framework. Although the resulting OpenCL code was particularly optimized for GPU architectures, it also runs on other devices (e.g. Intel Xeon Phi).

3.1 Considered parallelization models

When looking at the diversity of computational platforms that are commonly available today, two distinct QM/MM simulation cases deserve particular attention: running fewer Markov chains than the number of available OpenCL accelerators, and the opposite case. While the former is typically found in multi-node computational clusters, requiring the use of specially tailored communication-aware load balancing approaches (Li and Lan, 2005; Meng et al., 2010; Zheng et al., 2010), the latter is more common in single node systems. In this subsection, the available parallelism of both simulation environments is analyzed, by focusing on the simultaneous exploitation of multiple levels of

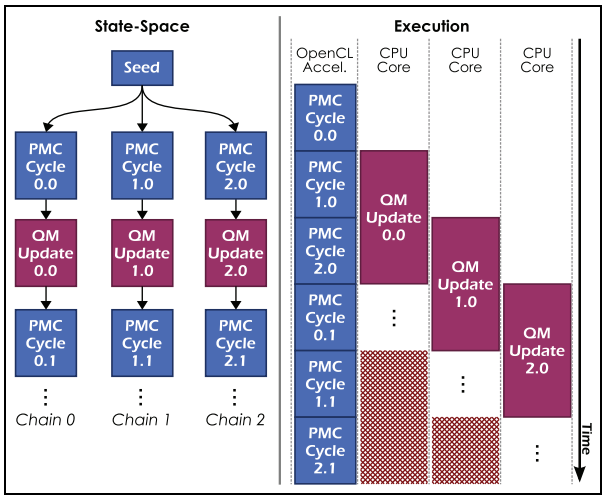


Figure 4. MC state-space alongside with the execution timeline for three Markov chains.

parallelism, in order to achieve a scalable solution for heterogeneous platforms.

The MC state-space can be sampled by running several Markov chains in parallel, allowing the simultaneous execution of the respective PMC cycles. Furthermore, this technique also allows execution of the QM update process for several chains in parallel, by using the available CPU cores. This execution layout is shown in Figure 4. Although the depicted example corresponds to three independent chains, this number can scale with the available computational resources. Moreover, to minimize the execution time, more OpenCL accelerators can be added to the system, by running more chains in parallel. Accordingly, the number of independent chains n that should be spawned in parallel in order to minimize the average per-chain execution time is dependent on: (i) the time taken for one CPU core to perform a chain QM update (t_{QM}); (ii) the time taken for each accelerator x to compute a chain PMC cycle (t^x_{PMC}), which is also dependent on the number of PMC cycle steps K (see Figure 2); and (iii) the number of CPU cores (C) and accelerators (A), and eventually limited by the amount of host memory. In particular, for cases where the host CPU cores represent the most limiting factor (i.e. $t_{QM}/C \gg t_{PMC}/A$), the number of independent chains should be set greater than the number of CPU cores ($n>C$) in order to guarantee a convenient overlap between QM and PMC cycle updates. On the other hand, whenever the system is limited by the time taken for the accelerators to perform the PMC cycle (i.e. $t_{QM}/C \ll t_{PMC}/A$), the optimal number of chains is mostly dependent on the number of accelerators, which should be set such that $n>A$. In other cases, the minimum per-chain execution time is usually attained when the number of chains is greater than the sum of the number cores and the number of accelerators ($n>C + A$).

On the other hand, the PMC cycle offers many opportunities for data and task-level parallelism. Figure 5 (left) shows the data dependencies of each process in the PMC cycle. In particular, the data structure corresponding to the changed molecule (*chmol*) is written by the Monte Carlo step and it is subsequently read by all the energy calculation procedures, which compute their respective ΔE energy terms to be processed by the Decide & Update procedure. Then, if the step under consideration is accepted, the *lattice* corresponding to the MM region (see Figure 1) is updated with the tested *chmol* configuration, and a new Monte Carlo step may take place. Unlike the other data structures, the *grid* corresponding to the QM region (see Figure 1) is not modified within the PMC cycle. Instead, it is updated by the QM update process. Hence, the described data dependencies within the PMC cycle imply that the energy contribution procedures can be executed in parallel with respect to each other. Furthermore, even the energy calculations are amenable to parallelism, as each of them can be mapped to a parallel reduction structure. For the particular case of the Coulomb grid QM/MM procedure, this can be verified by inspecting Algorithm 1. Having this in mind, the next subsection describes the implemented OpenCL kernels for the PMC cycle.

3.2 OpenCL kernels for the PMC cycle

The implemented OpenCL kernels for the PMC cycle are listed and mapped to the corresponding procedures in Figure 5 (right). The amount of parallelism that can be extracted within each kernel varies according to existing data dependencies and the amount of input data. Accordingly, it is highest in the *q3m_c* kernel, not only because the Coulomb QM/MM energy interaction is highly data-parallel (as can be observed by analyzing Algorithm 1), but also due to the size of the *grid* it takes as input, which may vary from hundreds of thousands to millions of grid points. This data set remains resident in the accelerator's memory between PMC cycle steps.

A diagram of the *q3m_c* and *q3m_finish* kernels is presented in Figure 6. Firstly, each work-item loads P grid points and the A atoms that comprise the *chmol* molecule from the global memory. The latter corresponds to a global memory broadcast, whereas the former is performed by P coalesced global memory read instructions, each reading a contiguous stripe of grid points to the work-group (step 1, in Figure 6). Then, for each {*atom, grid point*} pair, the corresponding work-item computes the squared Cartesian distance (within a periodic box) and compares it with squared cutoffs, thus avoiding an expensive *sqrt* operation. Depending on the resulting distance, the corresponding energy expression is computed (see the cutoff branches in Algorithm 1) and the results are accumulated in private memory (step 2). After this, the work-items of the

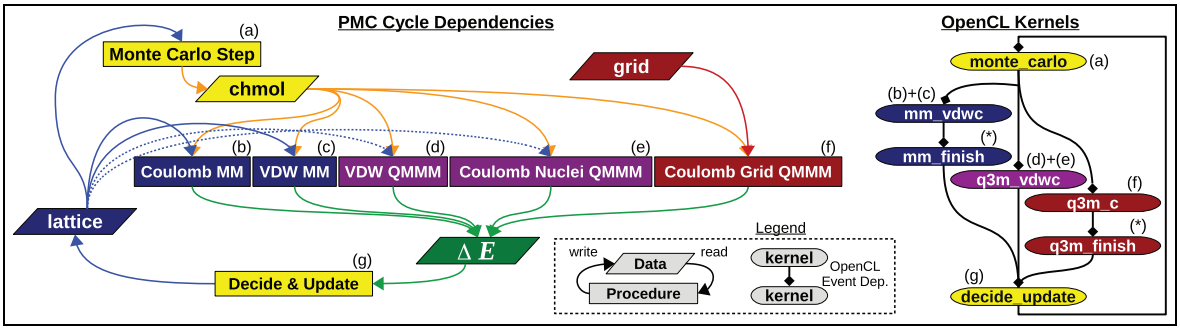


Figure 5. Data dependencies within the PMC cycle (left), together with a mapping of the PMC cycle procedures to OpenCL kernels (right). Kernels marked with a ($*$) correspond to auxiliary routines, only present in the parallel version. The *lattice* and *grid* data structures hold the data corresponding to the MM and QM regions, respectively (see Figure 1). Dotted lines represent partial reading of a data-structure.

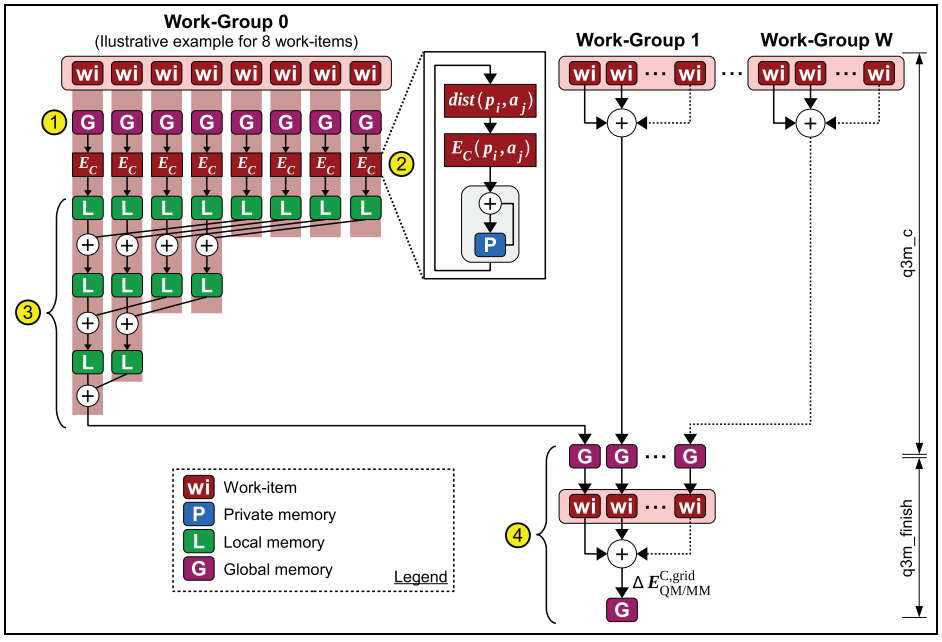


Figure 6. *q3m_c* and *q3m_finish* kernels structure. In this example, work-group 0 was presented with additional detail, although all work-groups share an identical structure. Likewise, the 8 work-items per work-group configuration was adopted for simpler illustrative purposes, as the work-group size is fully parametrizable.

same work-group reduce the computed energies in their local memory, by accumulating all terms in one single memory address after log_2(work − groupsize) iterations (step 3). Then, the first work-item of each work-group writes the obtained partial result in the global memory, and a final reduction kernel with only one work-group is launched (step 4), to reduce the remaining terms into a single value (different work-groups cannot communicate via global memory).

Hence, by including a first set of energy reductions in the same kernel as the $\Delta E^{C}_{QM/MM}$ energy computation (*q3m_c*), expensive global memory transfers that would otherwise be required between kernel launches are avoided. Furthermore, all reductions are organized in order to favor warp/wavefront release, ensuring that half of the active work-items finish their execution soon after each reduction iteration, thus promoting higher GPU occupancy. The corresponding reduction structure is presented in Algorithm 2, which closely resembles the recursive halving algorithm widely used in MPI applications (Thakur and Gropp, 2003).

The *mm_vdwc* and *q3m_vdwc* kernels have a fairly similar structure, except for the involved data and the considered energy expression. The former accounts for the Coulomb and vdW interactions between the *chmol* and the *lattice*, whereas the latter accounts for the interaction between the *chmol* and the QM atomic nuclei.

3.3 Exploiting multiple OpenCL devices

The two kernels that are responsible for the $\Delta E^{C,\,grid}_{QM/MM}$ energy contribution (*q3m_c* and *q3m_finish*) have the

Algorithm 2 Pseudo code for energy reduction.

```
Init: : local_size = Sizeofthiswork − group
Init: : local[local_id] = PrivateΔE^C_QM/MM energy
  for offset = local_size/2; offset>0; offset>> = 1 do
if local_id < offset then
    local_id<offsetlocal[local_id] = local[local_id +
    offset] + local[local_id];
      Local Barrier. Wait for work–group.
    end if
  end for
```

highest execution time in the original serial run (see Coulomb grid QM/MM term in Figure 2). Moreover, according to the dependency chart depicted in Figure 5, they only depend on the *chmol* data structure from the MC step and on the *grid* data (which is written once to the OpenCL device at the start of each PMC cycle). On the other hand, typical grids have hundreds of thousands to millions of points, allowing for a fine-grained partition among devices. Accordingly, all those conditions make these two kernels the best candidates for multi-device acceleration.

Figure 7 illustrates the described parallelization approach for a generic heterogeneous system composed of a host CPU and N different OpenCL devices. In this approach, the host is responsible for syncing the operations between the OpenCL devices, which share partial energy results on every iteration. In this particular example, device 0 is running all kernels, although *q3m_c* and *q3m_finish* only compute part of $\Delta E^{\mathrm{C,grid}}_{\mathrm{QM/MM}}$. Devices 1 to N, which might be accelerators with different compute capabilities, calculate the remaining terms of $\Delta E^{\mathrm{C,grid}}_{\mathrm{QM/MM}}$. The relative performance of the accelerators, together with the complexity of the assigned kernels, will determine the fraction of the *grid* that each one gets (G^0% to G^N%). More performance details for this type of configuration are presented in Section 4.

The overhead introduced by the device synchronization, to be executed at every step, is caused by several factors. Firstly, to read and write the partial energies of each device, one has to call the OpenCL functions *enqueueReadBuffer* and *enqueueWriteBuffer*, which also include an implicit *clFinish* statement to wait for the previous kernels in that step to finish (launches are chained using OpenCL events). This is accounted for in the *R/W Launches* block, in Figure 7. Secondly, each memory transfer introduces a small overhead corresponding to a copy of one floating point number per reduced energy term. Finally, syncing the host-side threads that are managing the OpenCL accelerators (*Barrier Sync*) and launching and parameterizing the OpenCL kernels (*Launch overhead*) also introduces some overhead.

3.4 Load balancing

To account for the possible heterogeneity of the computational platform, load balancing techniques were conveniently devised to distribute the computation of the system energy among the available OpenCL accelerators. In this respect, several strategies could be employed, including task-level approaches to distribute the computation of the several energy terms among the co-existing accelerators, data-level approaches to distribute the computation of each individual energy term, or mixed approaches. However, given the substantial difference between the computational cost of the Coulomb grid QM/MM energy term ($\Delta E^{\mathrm{C,grid}}_{\mathrm{QM/MM}}$) and the remaining terms (which account for less than 10% of the accelerated execution time), only data-level load balancing approaches are herein considered, such as distributing the computation of the Coulomb grid QM/MM energy term among the existing accelerators, as illustrated in Figure 7.

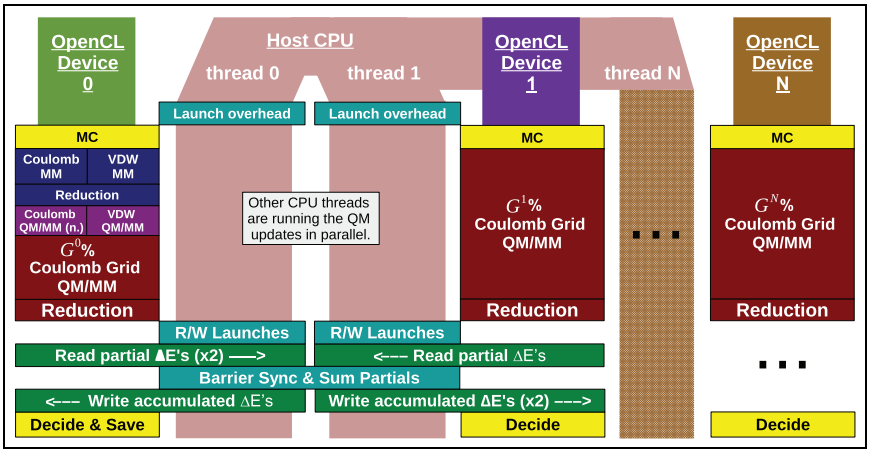


Figure 7. Exploiting multiple heterogeneous OpenCL devices to execute the PMC cycle. The execution is balanced by running different kernels on each device and dividing the work of the heavier kernels (*q3m_c* and *q3m_finish*).

The amount of *grid* data that is assigned to each device on each iteration is chosen according to a dynamic load balancing algorithm (adapted from Clarke et al. (2011)), by using performance information from previous iterations to balance the load. To accomplish this, the 3D *grid* is divided into n small and independent grid blocks. In the first step, all p devices are assigned the same number of blocks $d_i^0 = n/p$. Then, this distribution is conveniently updated every r steps. Thus, at step k, device 1 computes the grid blocks $b_1, \ldots, b_{d_1^k}$, device 2 computes blocks $b_{d_1^k+1}, \ldots, b_{d_1^k+1+d_2^k}$, and so on. All devices have access to all *grid* points, so that data displacement is not required.

Let $t_i(d_i^k)$ be the time taken by device i to compute the assigned d_i^k blocks (plus the remaining kernels it has been assigned to) in iteration k. The implemented load balance works as follows.

1. If $\max_{\text{all device pairs } (i,j)} \left| \frac{t_i(d_i^k) - t_j(d_j^k)}{t_i(d_i^k)} \right| < \epsilon$, the load is balanced. Skip step (2).
2. Recompute the amount of assigned grid blocks: $d_i^{k+1} = n \times \frac{d_i^k}{t_i(d_i^k) \times \sum_j^p \frac{d_j^k}{t_j(d_j^k)}}$

At this point, it is important to recall (see Section 2) that different cutoff regions may result in different energy expressions (or no energy computation at all). Consequently, the accelerators might compute over grid partitions that fall into different cutoff regions, thus having different computational efforts. Hence, the execution time measurements should be computed by averaging the execution time in several previous iterations, in order to avoid any misclassification of device performance.

The described algorithm is run by the host once every N iterations, after the *Barrier Sync* (see Figure 7). To accomplish this, the performance measurements of each device are shared between the corresponding host-threads, via host shared-memory.

4 Performance evaluation

4.1 Benchmarking setup

To experimentally evaluate the proposed parallelization approach, two sets of benchmarks were run on several hardware configurations. The first set includes benchmarks *bench-A, bench-B* and *bench-C* (see Table 1), which represent typical QM/MM setups and can therefore be used to assess the performance of the most demanding simulation (i.e. the PMC cycle), by running 10k steps. The QM part of these benchmarks consists of a set of protonated arginines that are acylated at the N-terminus and methylaminated at the C-terminus. This amino acid was solvated in a periodic water box (MM part), containing a variable number of water molecules (depending on the considered benchmark). The grid for the electronic charge density description was constructed by following Mura and Knowles ($\alpha = 1$ and $m = 3$) for the radial distribution (Mura and Knowles, 1996) and Lebedev ($l_{\max} = 53$) for the angular distribution (Lebedev, 1975). The QM calculations used the density functional PBE (Perdew et al., 1996) and the basis set def2-SVP (Weigend and Ahlrichs, 2005), while the MM part was described with the OPLS-AA force field (Jorgensen et al., 1996). Furthermore, the latest development version of the MOLPRO (Werner et al., 2012) program package (version 2012.1) was used in the QM calculations.

Bench-R consists of a smaller simulation box, designed for a much longer and realistic run. This simulation corresponds to the chorismate molecule in solution. Its conversion to prephenate is a widely studied biochemical reaction, as the same mechanism is followed in solution and in the chorismate mutase enzyme. The chemical aspects of this reaction are described in Claeyssens et al. (2011). For this benchmark, the run is comprised of 24.8 million steps, with the QM update executed every 50k steps (totaling 496 PMC outer iterations).

The considered hardware for the experimental setup is listed in Table 2. Different work-group partitioning schemes were used for each device. For NVIDIA GPUs, the CUDA calculator proved to be a useful tool for choosing starting point parameters. For AMD cards and Intel CPUs, the optimal values were found through test and experimentation, resulting in small multiples (e.g. 1 to 4) of the preferred elementary work-group size returned by an OpenCL device discovery query, made in runtime to the underlying platform.

Most of the presented performance comparisons are relative to the baseline version of the code executed on

Table 1. Considered QM/MM benchmark datasets. The chemical aspects of *bench-R* are presented in detail in Claeyssens et al. (2011).

Name	MM part	QM part	Grid size	MC steps
bench-R	500 H_2O molecules	Chorismate	183,356	24.8×10^6
bench-A	1301 H_2O molecules	1 Arginine	1,772,972	10×10^3
bench-B	5000 H_2O molecules	1 Arginine	1,772,972	10×10^3
bench-C	5000 H_2O molecules	2 Arginines	2,992,458	10×10^3

Table 2. Computational platforms considered in the experimental evaluation.

Platform	Host				OpenCL accelerators	
	CPU	Freq.	#Cores	RAM	Devices	RAM
$mA_{i7-4770K}$	Intel Core i7-4770K	3.5 GHz	4	32 GB	-	-
$mA_{i7-4770K}^{GTX780}$	Intel Core i7-4770K	3.5 GHz	4	32 GB	NVIDIA GTX 780Ti	3 GB
$mA_{i7-4770K}^{GTX780,GTX660}$	Intel Core i7-4770K	3.5 GHz	4	32 GB	NVIDIA GTX 780Ti/660Ti	3 GB/2 GB
$mA_{i7-4770K}^{i7-4770K}$	Intel Core i7-4770K	3.5 GHz	4	32 GB	i7-4770K	32 GB
$mB_{i7-3820}^{R9-290X,GTX560}$	Intel Core i7-3820	3.6 GHz	4	16 GB	AMD R9 290X/NVIDIA 560Ti	3 GB/1 GB
$mC_{E5-2609}^{GTX680,GTX680}$	2x Xeon E5-2609	2.4 GHz	2x4	32 GB	2x NVIDIA GTX 680	4 GB/4 GB
$mD_{i7-3770K}^{K20c}$	Intel Core i7 3770K	3.5 GHz	4	8 GB	NVIDIA Tesla K20C	5 GB

a single core of the i7-4770K processor, compiled with Intel compiler (ICC v13.1.3) with flag-*fast*, such as to use AVX2 instructions on all *mA* platforms and AVX instructions in the remaining platforms. Furthermore, since the CPU with higher performance is installed in platform $mA_{i7-4770K}$, this will be used as baseline for all performance comparisons (using a single core), unless otherwise specified. The newest available OpenCL standard was used for each device (OpenCL 1.1 for the considered NVIDIA GPUs and OpenCL 1.2 for the Intel CPUs and AMD GPUs). Furthermore, both double-precision (fp_{64}) and mixed double and single-precision ($fp_{64} - fp_{32}$) data-types will be employed in the performance evaluation performed in this section. Details about these numerical configurations and the corresponding compromises, as well as a mixed fixed-point precision approach, will be discussed in Section 5.

Table 3. Total execution time (T_{all}) and time for output writing (T_{out}) for a PMC cycle with 10k iterations, in several hardware platforms, when using $fp_{64} - fp_{32}$ mixed-precision. All execution times are presented in seconds.

Platform name	Benchmark		
	bench-A	bench-B	bench-C
$mA_{i7-4770K}$	T_{all} = 769.96	787.68	1179.70
	T_{out} = 0.23	0.76	0.76
$mA_{i7-4770K}^{i7-4770K}$	T_{all} = 137.90	140.25	232.20
	T_{out} = 0.90	6.75	6.87
$mA_{i7-4770K}^{GTX780}$	T_{all} = 6.33	8.08	11.20
	T_{out} = 0.53	1.70	1.72
$mA_{i7-4770K}^{GTX780,GTX660}$	T_{all} = 5.04	6.73	8.68
	T_{out} = 0.52	1.66	1.66
$mB_{i7-3820}^{R9-290X,GTX560}$	T_{all} = 6.90	8.42	11.30
	T_{out} = 0.57	2.02	2.02

4.2 PMC cycle acceleration

Table 3 presents the PMC cycle execution time (10k steps) obtained for benchmarks *bench-A, bench-B* and *bench-C*, when profiled for several hardware configurations. The overall execution time corresponds to the cost of running 10k steps, plus the final output flushing from the OpenCL device(s) to the host, and file writing (*Output* time in Table 3, ranging from ~0.5 s to ~2 s). The extra overheads related to the OpenCL initialization time and the input file reading were not accounted for, because they do not scale with the simulation size and would be diluted in longer runs (contrary to output generation).

The difference between the execution times of *bench-A* and *bench-B* corresponds to the number of MM molecules. This introduces two implications, namely, *bench-B* imposes a heavier footprint of the *mm_vdwc* and *mm_finish* kernels, and an increased size of the generated output, which in turn means a heavier *decide* kernel and a longer output flushing. The latter can be observed in Table 3 and mainly depends on the host-to-device communication speed to read the output, and on the time to write the output file.

On the other hand, *bench-C* has a larger QM part, resulting in heavier *q3m_c* and *q3m_finish* kernels. This favors the overall performance with respect to *bench-B*, as the performance of the most data-parallel kernels is favored by a higher number of *grid* points. The speed-up results of the parallel platforms with respect to the $mA_{i7-4770K}$ (and corresponding to the execution times presented in Table 3) are depicted in Figure 8.

According to the presented results, the speed-up values in the PMC cycle acceleration are fairly high when compared to $mA_{i7-4770K}$. This is a direct consequence of a careful exploitation of the memory hierarchy, together with the higher memory bandwidth in GPU architectures. In fact, although CPUs compensate the lower main memory bandwidth with multiple levels of caches, the most intensive procedure in the PMC cycle (Coulomb grid QM/MM) requires loading a huge amount of data from main memory at each step (e.g. up to 48 MB for the case of *bench-C*), rendering the first cache levels useless. Nevertheless, coalesced memory accesses still exploit parallelism when accessing the main GPU device memory, regardless of using local caches or not.

Table 4. Kernel execution times in platform $mA_{i7-4770K}^{GTX780}$ for the particular case of *bench-A*.

Kernel name	Performance		Speed-up
	i7-4770K	GTX 780Ti	
monte_carlo	*32 μs (0.04%)*	*17 μs (3.0%)*	1.88×
q3m_c/reduce	*76,077 μs (98.8%)*	*473 μs (83.3%)*	160.84×
mm_vdwc/reduce	*791 μs (1.03%)*	*40 μs (7.0%)*	19.77×
q3m_vdwc	*4 μs (0.01%)*	*18 μs (3.2%)*	0.22×
decide	*94 μs (0.12%)*	*20 μs (3.5%)*	4.70×
Total	**76,998 μs (100%)**	**568 μs (100%)**	135.55×

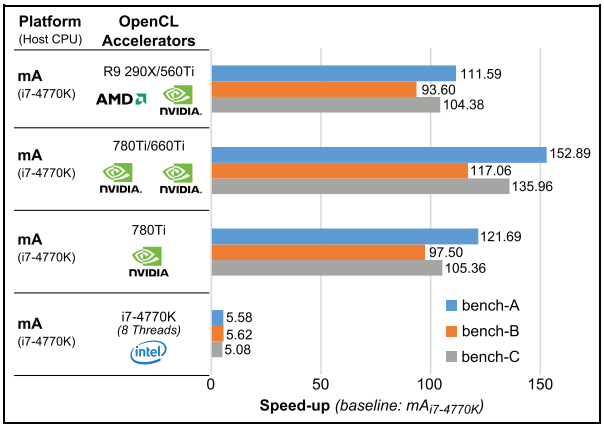


Figure 8. Speed-up obtained for a PMC cycle with 10k iterations, when using $fp_{64} - fp_{32}$ mixed-precision. The corresponding execution times are presented in Table 3.

Table 4 presents the kernel execution times for the particular case of the *GTX780Ti* accelerator, together with the times corresponding to the reference implementation in the $mA_{i7-4770K}$ platform. As can be observed, the kernels that achieve the highest speed-up are the *q3m_c* and *q3m_finish*, as predicted in Section 3.2. The very large speed-up attained in these kernels (160.84×) is subsequently affected by Amdahl's law (considering the fractions and speed-ups of all the other kernels) and results in an overall PMC cycle step speed-up of 135.55×. By recalling the execution times presented in Table 3 for the particular case of the *GTX780Ti* accelerator, the resulting speed-up without considering the *Output* overhead would be $\frac{769.96-0.231}{6.33-0.534} = \sim 132.8\times$ (versus the 121.29× value, presented in Figure 8, where every component is taken into account), which is slightly below the speed-up attained in the PMC cycle step, due to device management and kernel launching overheads, not accounted for in Table 4.

4.2.1 PMC cycle load balancing. Figure 9 presents the kernel timing results per PMC cycle step, when considering *bench-A* executing on the $mA_{i7-4770K}^{GTX780}$ heterogeneous platform. The load balancing algorithm introduced in Section 3.4 was used and converged to the *grid* partitioning depicted in this figure. Figure 10 illustrates the workload balancing evolution with time. Here, the

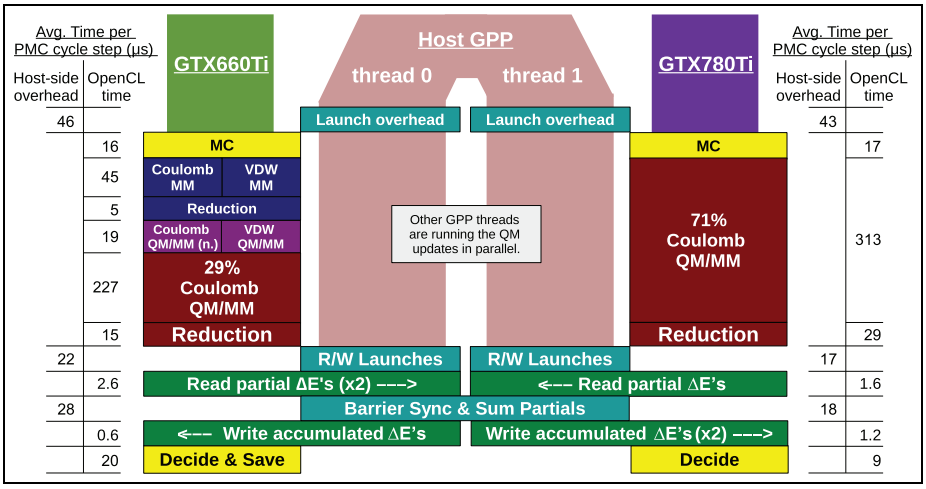


Figure 9. OpenCL kernel timings (per step) for the PMC cycle on the $mA_{i7-4770K}^{GTX780,GTX660}$ heterogeneous platform. The load is balanced for the heavier kernels (*q3m_c/q3m_finish*, corresponding to Coulomb QM/MM), whereas the lighter kernels were scheduled to the first GPU. The considered benchmark is *bench-A*, using the mixed $fp_{64} - fp_{32}$ precision.

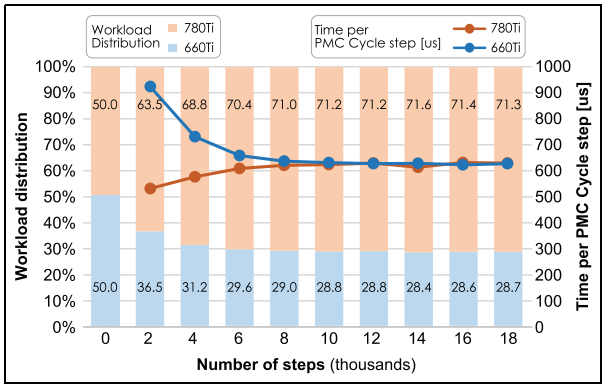


Figure 10. Convergence pattern of the implemented load balancing algorithm (balancing every 2000 steps), for *Bench-C* running on the platform $mA_{i7-4770K}^{GTX780,GTX660}$. The presented PMC cycle time measurements represent mean times since the previous balancing.

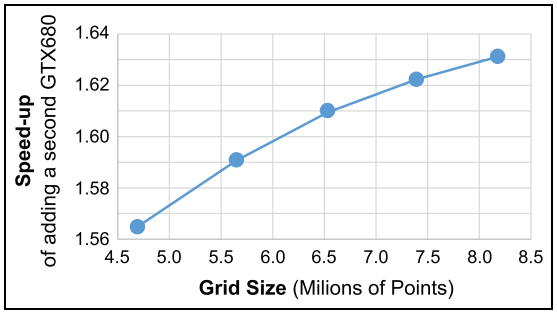


Figure 11. Scalability of the PMC cycle on platform *mC* when changing the size of the QM part in *bench-A*. Speed-up results are presented for a dual *GTX680* system with respect to a single *GTX680*.

balancing term r is set to 2000 iterations, in order to avoid under-sampling the computational weight of the *q3m/mm* kernels (which depends on the randomly picked MM molecule). The starting workload distribution of 50%/50% converges to approximately 71%/29% in only 4 balancing steps, favoring the more powerful 780Ti GPU. When this distribution is reached, one can observe that the execution of the balanced workload in each GPU takes approximately the same time, which means that the load is balanced and that the balancing mechanism has met its purpose. In order to illustrate how the balancing persists even after the 10k−th run, the chart represents the execution up to 20k steps. When compared with an unbalanced run (e.g. fixed 50%/50% workload distribution) on the same platform, the balanced version yields a speed-up of 1.3×, further justifying the advantage of having incorporated a load balancing solution.

4.2.2 PMC cycle scalability. The memory footprint of the PMC cycle kernels in the OpenCL accelerators is mainly limited by the program output buffers, the pre-generated random lists required for the *monte_carlo* kernel, the MM lattice and the QM grid. The first two solely depend on the number of executed steps, and are addressed by having the host CPU flushing the output and refreshing the random lists periodically. The second two were also not a problem for the selected benchmarks, since the largest used QM grid and MM lattice occupy ~48 MB and ~160 KB, respectively.

Scheduling the computations belonging to a single Markov chain among multiple GPUs comes with an additional overhead of syncing the PMC cycle step results among the involved devices, at the end of every step. Fortunately, these overheads do not scale with the simulation size, since the buffers that need to be synced back and forth (between the host and the accelerators) are reduced energy terms, each represented by a single number. For the particular example presented in Figure 9, each device has to read/write three reduced terms per step, which has a performance impact of a few dozen microseconds. Conversely, the computational cost of the *q3m_c/reduce* kernels scales with the size of the QM *grid*, meaning that multi-device scalability is better for larger grids (which concern the most computationally challenging problems).

Figure 11 presents the obtained speed-ups for the PMC cycle kernels acceleration, when a second GPU is added to platform $mC_{E5-2609}^{GTX680,GTX680}$ to balance the same Markov chain, by considering several QM grid sizes. As can be observed, multi-device performance scales better for simulations with greater QM regions, which actually represents a common characteristic of real QM/MM systems. Considering that the high speed-up results obtained in the *q3m_c/reduce* kernels, one might expect the execution time of these kernels to slowly increase with the introduction of larger grids, justifying the slow scaling of the dual-device speed-up curve presented in Figure 11. Nevertheless, it is important to recall that the execution time of the PMC cycle kernels (for each Markov chain) was already reduced to the micro-second order of magnitude for a single device, and that at this level, every overhead is noticeable. Therefore, the observed dual-device speed-up, ranging from ~1.5× to ~1.6×, is deemed favorable for *grids* up to 8 million points. This speed-up would continue to rise (never exceeding 2×) for larger and more computationally intensive QM/MM systems.

4.3 Global PMC results

To conclude this evaluation of the proposed parallel solution, the execution of the complete PMC simulation is assessed (including the QM update and the PMC cycle stages). For such a purpose, a greater focus will be given to *bench-R*, corresponding to the longest and more realistic dataset. Figure 12 depicts the simulation

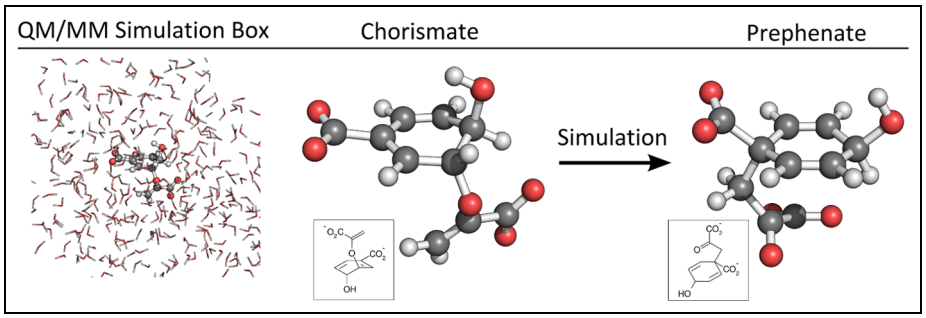


Figure 12. QM/MM Simulation box for the *bench-R* dataset (partial representation), together with the simulation results for the conversion of the chorismate structure into prephenate.

Table 5. *bench-R* execution time for the PMC cycle (50k steps) and QM update (24.8M iters) stages, as well as for the full PMC simulation.

Platform	Format	Num. Markov chains	PMC cycle		QM upd.	Full simulation	
			Time (s)	Speed-up vs serial	Time (s)	Time (s)	Speed-up vs serial
$mC_{E5-2609}$ *(serial with AVX)*	fp_{64}	1	1883.1	1×	96.4	980038.0	1×
$mC^{GTX680,GTX680}_{E5-2609}$	fp_{64}	1	42.9	43.80×	96.4	69026.4	14.20×
		8	42.9	43.80×	113.4	10156.2	96.50×
$mC^{GTX680,GTX680}_{E5-2609}$	fp_{64}-fp_{32}	1	10.2	184.23×	96.4	52833.4	18.55×
		8	10.2	184.23×	113.4	7757.7	126.33×

Table 6. *bench-R* execution time and speed-up when comparing the baseline $mCfp_{64}$-fp_{32} configuration running either 1 or 8 Markov chains and $mA_{i7-4770K}fp_{64}$ configuration with 1 or 8 Markov chains (same conditions as in Table 5).

Platform	Num. Markov chains	PMC cycle		Full simulation	
		Time (s)	Speed-up	time (s)	Speed-up
$mA_{i7-4770K}$	1	572.7	1×	300028.0	1×
$mC^{GTX680,GTX680}_{E5-2609}$	1	10.2	56.02×	52833.4	5.67×
$mC^{GTX680,GTX680}_{E5-2609}$	8	10.2	56.02×	7757.7	38.68×

results, showing the conversion of the chorismate structure into prephenate.

Table 5 presents the execution times for the inner PMC cycle (comprising 50k steps), the QM update and the full PMC application (comprising 496 PMC outer iterations, which yields a total of 24.8 M PMC cycle steps). This performance study was conducted with the $mC^{GTX680,GTX680}_{E5-2609}$ platform (2x Intel Xeon E5-2609), since it has the largest number of CPU cores (8), allowing to spawn up to 8 independent Markov chains while scheduling their respective PMC cycles on two GTX680 GPUs. The execution times were measured for a single and for 8 Markov chains. The same experiment was also conducted on a single-core of this $mC^{GTX680,GTX680}_{E5-2609}$ platform. In order to better evaluate the presented parallelization efficiency, the performance was also measured in the $mA_{i7-4770K}$ platform (single core, with AVX2), which was considered as the *reference* platform in the conducted comparisons. From the presented results, it can be observed that although the CPU cores in this reference platform are faster than the ones in platform $mC^{GTX680,GTX680}_{E5-2609}$ (running the QM update roughly 3× faster), considerable speed-up gains can still be achieved by offloading the computation of the PMC cycle to the NVIDIA GTX680 GPUs, as presented in Table 6.

When considering all these configurations, it is observed that the execution time of the full *bench-R* for the $mC^{GTX680,GTX680}_{E5-2609}$ reference scenario corresponds to 272.23 h (980038 s), while it takes 83.34 h (300028 s) in the $mA_{i7-4770K}$ platform. The parallelized solutions reduce these execution times considerably, ranging from 19.17 h (69026.4 s) to 2.15 h (10156.2 s), depending on the number of spawned Markov chains (either

single or 8 chains) and the chosen numerical precision. For the single-chain case, the obtained speed-up is mainly due to the OpenCL acceleration of the PMC cycle. As shown in Table 5, a speed-up of up to 184.23× is obtained in the PMC cycle alone. However, this speed-up will be affected by Amdahl's law, due to the QM update fraction running on the CPU. In fact, by looking at the $mC_{E5-2609}^{GTX680,\,GTX680}$ reference scenario, one can observe that in the original run the PMC cycle represented $\frac{1883.1}{1883.1+96.4} = 95.13\%$ of each PMC iteration (PMC cycle + QM update). Hence, the speed-up of 18.55× presented in Table 5 was expected, since the speed-up of 184.23× obtained in the PMC cycle ($fp_{64}-fp_{32}$ version) would at maximum yield $\frac{1}{\frac{0.9513}{184.23}+0.0487} \approx 18.57\times$ global speed-up.

For the multiple Markov chain case, the attained speed-up is mainly due to the parallel MC state-space exploration. In fact, by comparing the single with the multiple chain speed-up values for the same precision approach, a scalable speed-up trend can be observed from the obtained results. For example, by comparing the speed-ups attained in the $mC_{E5-2609}^{GTX680,\,GTX680}$ platform ($fp_{64}-fp_{32}$ mixed precision) for the cases of 1 and 8 chains, a speed-up ratio of $\frac{126.33\times}{18.55\times} = 6.81\times$ is obtained. It is important to recall that the speed-up attainable by adding more chains is dependent on the number of CPU cores and accelerators, on the host-side thread management, and on the overhead introduced by concurrent memory and disk accesses issued by the CPU cores running the QM updates in parallel. In this case, one can verify from Table 5 that the QM update mean execution time has degraded from 96.4 s to 113.4 s, where the considered multiple Markov chain solution is achieved with an efficiency of $\frac{38.68}{5.67\times 8(\#\text{cores})} = \sim 85\%$, resulting from the existing contention and non-perfect overlapping between QM and PMC cycle updates.

Among the presented results, the most conservative speed-up (36.86×) of this parallel implementation is attained when comparing platform $mC_{E5-2609}^{GTX680,\,GTX680}$ ($fp_{64}-fp_{32}$ mixed-precision) running 8 chains versus $mA_{i7-4770K}$ (fp_{64} precision), the reference serial execution. Naturally, the 126.33× speed-up obtained when comparing $mC_{E5-2609}^{GTX680,\,GTX680}$ to itself could remain close to this value if a better Intel Xeon CPU had been used in both the reference and the parallel solutions.

5 Numerical evaluation: Convergence accuracy and energy consumption

The proposed parallel implementation does not make any approximation or relaxation with respect to the original sequential method, yielding exactly the same output. Moreover, besides the original 64bit floating-point representation (fp_{64}), the presented OpenCL version also offers the following numerical representation alternatives: i) mixed 64bit and 32bit floating-point ($fp_{64}-fp_{32}$); or ii) mixed 64bit/32bit floating-point and 32bit fixed-point ($fp_{32}-i_{32}$). In the $fp_{64}-fp_{32}$ version, the computationally more complex *q3m_finish*/*mm_finish* kernels use 32bit floating-point precision for the $\Delta E_{\text{QM/MM}}^{\text{C, grid}}$ energy computations, whereas 64bit floating-point is employed for the remaining energy terms, which have much faster computations. This configuration also assumes the same data-type to store the *grid*, as well as a copy of the *lattice* and the *chmol*. Likewise, the $fp_{32}-i_{32}$ version uses 32bit floating-point for the same energy computation, but it uses 32bit fixed-point for the squared distances. The latter operates on normalized *grid* and atom coordinates, represented by 32bit integers, which actually provides a higher precision than the alternative 32bit floating-point.

Table 7. Speed-up of the mixed precision *q3m_c* kernel versions versus the original fp_{64} version, running on the same machine, for the case of *bench-A*.

Format	q3m_c speed-up (vs f64)		
	GTX680	GTX780Ti	K20C
$fp_{64}-fp_{32}$	8.56×	7.39×	2.65×
$fp_{32}-i_{32}$	8.89×	7.44×	2.74×

Table 8. Obtained numerical precision for the $\Delta E_{\text{QM/MM}}^{\text{C}}$ energy term and the total energy of the system (E), when considering the complete set of QM/MM systems, generated using *bench-A*. The relative (percentage) values consider as baseline the maximum error $e_m = 1.0\times10^{-1}$ kJ/mol.

Format		Error vs fp_{64}	
		$\Delta E_{\text{QM/MM}}^{\text{C}}$(kJ/mol)	E(kJ/mol)
$fp_{64}-fp_{32}$	mean	6.4×10^{-5} (0.064%)	4.2×10^{-3} (4.2%)
	max	2.9×10^{-3} (2.9%)	1.6×10^{-2}
$fp_{32}-i_{32}$	mean	1.6×10^{-5} (0.016%)	9.0×10^{-4}(0.9%)
	max	1.1×10^{-3} (1.1%)	1.6×10^{-2} (16%)

The resulting performance gains, for each of the considered precisions when executing the *q3m_c* kernel, are presented in Table 7. Depending on the adopted GPU device, execution speed-ups as high as 8.89× can be attained by simply using lower resolutions, with minor degradations of the obtained energy results. However, the generated system configurations will be the same as long as the accumulated error does not cause the sequence of selected systems to diverge, which was verified to be the case for all the considered benchmarks. Table 8 presents the amount of error that is introduced in the $\Delta E_{\text{QM/MM}}^{\text{C, grid}}$ term for each kernel version and the total system energy (E), with respect to the fp_{64} implementation (which is numerically equivalent to the original serial version). It can be observed that the

Table 9. Execution time, speed-up, energy savings and average power consumption when running all the devised numerical precision approaches. The testbench was run for 100k steps, in order to ensure a representative sampling of the computational cost of the OpenCL accelerated *q3m_c* kernel. The default core frequency configuration was used for all experiments.

Platform	Format	*q3m_c* Kernel			
		Time (μs)	Avg. power	Speed-up	Energy savings†
$mA^{i7-4770K}_{i7-4770K}$	fp_{64}	13031	84 W	1×	-
	$fp_{64} - fp_{32}$	6157	76 W	2.12×	57%
	$fp_{32} - i_{32}$	6143	75 W	2.12×	58%
$mD^{K20C}_{i7-3770K}$	fp_{64}	1775	140 W	7.34×	-
	$fp_{64} - fp_{32}$	670	147 W	19.45×	60%
	$fp_{32} - i_{32}$	647	139 W	20.14×	64%

† Energy savings versus corresponding fp_{64} version

$fp_{32} - i_{32}$ version offers higher precision than the $fp_{64} - fp_{32}$, due to the greater number of significant bits used for the squared distances operations. In these simulations, it was verified that the error never exceeded $e_m = 1.0 \times 10^{-1}$kJ/mol, as commonly considered in this research domain.

In order to further assess the impact of the considered mixed precision solutions, the average energy consumption was measured on the NVIDIA Tesla K20C GPU, by using the NVML library. The method presented in Guerreiro et al. (2015) was used to gather the attained power measurements, by using the maximum allowed sampling frequency of 66.7Hz. Since this frequency is too low to sample one kernel launch of *q3m_c* (which executes in the order of hundreds of microseconds), a testbench with just the *q3m_c* kernel was built and launched repeatedly for 100k steps. The obtained results are presented in Table 9.

The first aspect worth noting refers to the configuration that presented the highest average power: the $fp_{64} - fp_{32}$. This fact can be justified by specific GPU architecture characteristics that allow attaining higher core occupancy with the single precision floating-point implementation. The fp_{64} version has a lower average power dissipation due to the opposite reason, i.e. its lower GPU occupancy, resulting in a reduced dynamic power requirement. For the case of the $fp_{32} - i_{32}$ configuration, a similar GPU occupancy relative to $fp_{64} - fp_{32}$ is expected, although the integer functional units consume less power, resulting in a 8 W decrease in average power. To complete these observations, power and energy consumption were also measured on the $mA^{i7-4770K}_{i7-4770K}$ OpenCL accelerated platform, using the power and energy measurement tool by Taniça et al. (2014). Although the fp_{64} configuration draws (on average) approximately 1.65 times less power than the most energy efficient parallel configuration on the NVIDIA Tesla K20C GPU ($fp_{32} - i_{32}$), the acceleration attained by the GPU in the execution time of the *q3m_c* kernel greatly compensates this, yielding a much lower overall energy consumption, thus attaining up to 64% energy savings. Although the same tests could not be performed on the GTX680 and GTX780Ti GPUs (they do not support NVML power readings), one can predict rather similar energy savings for the GTX780Ti GPUs accelerator, since it shares the same Kepler core architecture (GK110).

6 Related work

Similar research on molecular simulation acceleration that is found in the literature is typically classified in terms of: i) the nature of the employed sampling; ii) the theory used for the energy calculations; and iii) the chemical application for which they have been tuned. The employed sampling is usually performed in time (MD) or in state-space (MC) and the energy interactions may consider pure QM, pure MM and mixed QM/MM terms. Finally, the application for which the algorithm has been tuned to may vary greatly, and this is the main reason why the attained performance gains in the parallelization of these algorithms can seldom be compared to each other.

Although computationally demanding, MD is a popular approach to study a wide range of dynamical properties. Hence, several approaches were proposed for its acceleration, dating from the earlier times of GPGPU (Stone et al., 2007; Friedrichs et al., 2009) to more recent publications (Pratas et al., 2012; Mashimo et al., 2013; Nitsche et al., 2014). On the other hand, methods based on MC sampling allow simulating systems with longer timescales, and several works have also accelerated these algorithms with GPGPU (Anderson et al., 2007; Uejima et al., 2011; Anderson et al., 2013; Lutsyshyn, 2013; Hall et al., 2014). Since our work falls into the latter category (MC), a more detailed review of such works is presented.

In particular, Anderson et al. (2007) propose a GPGPU solution for quantum Monte Carlo (QMC), achieving up to 30× speed-up in individual kernels and up to 6× speed-up in the overall execution. The QMC variety that is considered by such research is based on

diffusion Monte Carlo (DMC), unlike the PMC approach followed in our work. They employ a scheme for simultaneous state-space exploration (each chain called a walker) by using a scheme similar to the multiple Markov chain approach that is herein adopted. However, they focus on exploiting parallelism at the walker level (up to 16 simultaneous walker evaluations on the GPU), whereas we focus on exploiting the finer-grain level of parallelism on each chain (which are heavy enough to keep the GPU busy, in our case), and manage chain-level parallelism with fewer chains per GPU. We took this approach since spawning a very large number of chains on the same GPU would be unfeasible for the case of the PMC method, since each chain requires computation not only of the MC step trials (in this case, the PMC cycle procedures), but also the intrinsically serial QM update process.

On the other hand, Uejima et al. (2011) use MC sampling (based on variational Monte Carlo) and target QM/MM systems. Just like the algorithm described in this work, the bottleneck arises in the computation of the electrostatic potential. They target computational clusters composed of GPUs to handle the bottleneck code and CPUs for the remaining procedures, obtaining a speed-up of up to 23.6× versus a single-core CPU. The adopted GPGPU framework is CUDA, and an MPI solution is proved scalable up to 4 CPU cores, when considering the Intel Core i7-920 architecture. They do not report any explicit load balancing solution nor target the simultaneous exploitation of heterogeneous GPU platforms, contrary to the work herein presented.

Anderson et al. (2013) describe a CUDA GPGPU implementation for many-particle simulations using MC sampling. They partition the particle set in several cells and apply many MC steps in parallel, known to not interfere with each other. They do not target QM/MM systems. Instead, tests are performed for a 'hard disk' system (two-dimensional particles which cannot overlap), and the considered particle interactions are the physical collisions. Unlike physical collisions, the electrostatic potentials considered in our work have a much higher range, and as such the computed energy terms at each MC step depend on a much larger number of their neighbor molecules (the potential cutoffs are about half of the simulation box). Hence, such a scheme would not effectively solve the problem that is herein considered, as most MC steps would interfere with each other. The work presented in Hall et al. (2014) also describes a parallel approach to particle MC simulations using CUDA, without any emphasis on QM/MM systems.

Finally, the work by Nitsche et al. (2014) targets QM/MM simulations, although time sampling (MD) is used instead, and a special focus is given to accelerating the QM grid generation, achieving up to 30× speed-up. This contrasts to what happens in the PMC, where the bottleneck is found in the QM/MM electrostatics (the PMC cycle), which is significantly accelerated by our implementation.

Also, it is worth recalling that direct performance comparisons are difficult to handle in this field, and very few authors do it in the literature. Furthermore, very few have considered the use of heterogeneous architectures for hybrid QM/MM simulations, whilst using MC sampling. The proposed solution efficiently takes advantage of the hybrid nature of QM/MM simulations and the MC state-space exploration, unlike typical pure QM or MM approaches.

Finally, it is worth noting that most existing works adopted CUDA as the programming framework, being constrained to NVIDIA GPUs. To circumvent this limitation, other frameworks have been developed to ease the programming of non-conventional architectures, such as StarPU (Augonnet et al., 2011) and OpenCL. Due to its simpler means to orchestrate multiple devices in an heterogeneous environment and to write portable code between different architectures, the latter was used in this work. Moreover, by allowing an easy extension with the MPI framework (e.g. SnuCL by Kim et al. (2012) extends the original OpenCL semantics to a CPU/GPU cluster environment, without any required modifications in the original OpenCL program), the proposed approach opens the possibility of attaining further performance scalability at the chain-level, since the most challenging fine-grained part consisting of the parallelization of the PMC cycle was already overcome.

7 Conclusions

An OpenCL implementation of a novel QM/MM Monte Carlo model was presented, specially geared towards chemical simulations in the condensed phase. It was shown that the use of a hybrid parallel architecture makes particular sense in this context, as one can exploit both the inherent parallelism in the interaction between the MM and QM systems, and the task and data parallelism available in the involved computations. Concurrently to the OpenCL accelerators, the host CPU manages workload distribution in order to ensure a dynamic load balancing, and executes the QM grid update.

The devised solution proved to be particularly well suited for heterogeneous systems. Since pure QM codes are not as amenable to parallelism as force field calculations, the QM update is especially fit for execution on host CPU, allowing the more time-consuming PMC cycle to be executed on the OpenCL accelerators. It was also verified that the efficiency and scalability of the proposed solution on several different heterogeneous platforms is composed of multi-core CPUs and multiple and very different GPUs. As a result, by exploiting the

massively parallel GPU architecture, the computational bottleneck in the original single-core approach was accelerated by 56 ×, for the case of a well known chorismate dataset. To further promote the scalability of the proposed implementation, the MC state-space was further sampled using several independent Markov chains, which was proved to scale with an efficiency of 85%. In a commutative perspective, the complete PMC simulation yielded a speed-up of 38 ×, with particularly favorable scalability for many-node CPU clusters equipped with OpenCL accelerators. As a result, the devised solution effectively reduced the whole execution time of the chorismate QM/MM simulation from ~80 hours to ~2 hours, representing considerable savings in terms of time and energy.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under projects Threads (ref. PTDC/EEA-ELC/117329/2010), P2HCS (ref. PTDC/EEI-ELC/3152/2012) and project UID/CEC/50021/2013.

References

Anderson AG, Goddard WA III and Schröder P (2007) Quantum Monte Carlo on graphical processing units. *Computer Physics Communications* 177(3): 298–306.

Anderson JA, Jankowski E, Grubb TL, et al. (2013) Massively parallel Monte Carlo for many-particle simulations on GPUs. *Journal of Computational Physics* 254: 27–38.

Augonnet C, Thibault S, Namyst R, et al. (2011) StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23(2): 187–198.

Claeyssens F, Ranaghan KE, Lawan N, et al. (2011) Analysis of chorismate mutase catalysis by QM/MM modelling of enzyme-catalysed and uncatalysed reactions. *Organic and Biomolecular Chemistry* 9(5): 1578–1590.

Clarke D, Lastovetsky A and Rychkov V (2011) Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms. *Parallel Processing Letters* 21(02): 195–217.

Fennell CJ and Gezelter JD (2006) Is the Ewald summation still necessary? Pairwise alternatives to the accepted standard for long-range electrostatics. *The Journal of Chemical Physics* 124(23): 234104. DOI: 10.1063/1.2206581.

Friedrichs MS, Eastman P, Vaidyanathan V, et al. (2009) Accelerating molecular dynamic simulation on graphics processing units. *Journal of Computational Chemistry* 30(6): 864–872.

Geromichalos GD (2007) Importance of molecular computer modeling in anticancer drug development. *Journal of B.U.ON.: Official Journal of the Balkan Union of Oncology* 12(Suppl 1): 101–118. Available at: http://europepmc.org/abstract/MED/17935268.

Guerreiro J, Ilic A, Roma N, et al. (2015) Multi-kernel autotuning on GPUs: Performance and energy-aware optimization. In: *23rd Euromicro international conference on parallel, distributed and network-based processing (PDP)*, Turku, Finland, 4–6 March, pp.438–445.

Hall C, Ji W and Blaisten-Barojas E (2014) The metropolis Monte Carlo method with CUDA enabled graphic processing units. *Journal of Computational Physics* 258: 871–879.

Jorgensen WL, Maxwell DS and Tirado-Rives J (1996) Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids. *Journal of the American Chemical Society* 118(45): 11225–11236.

Kim J, Seo S, Lee J, et al. (2012) SnuCL: An OpenCL framework for heterogeneous CPU/GPU clusters. In: *Proceedings of the 26th international conference on supercomputing (ICS'12)*, Venice, Italy, 25–29 June, pp.341–352.

Lebedev V (1975) Values of the nodes and weights of ninth to seventeenth order Gauss-Markov quadrature formulae invariant under the octahedron group with inversion. *USSR Computational Mathematics and Mathematical Physics* 15(1): 44–51.

Li Y and Lan Z (2005) A survey of load balancing in grid computing. In: Zhang J, He J-H and Fu Y (eds) *Computational and Information Science: First International Symposium, CIS 2004*, Shanghai, China, 16–18 December, pp. 280–285. DOI: 10.1007/978-3-540-30497-5_44.

Lutsyshyn Y (2015) Fast quantum Monte Carlo on a GPU. *Computer Physics Communications* 187: 162–174. DOI: 10.1016/j.cpc.2014.09.016.

Mashimo T, Fukunishi Y, Kamiya N, et al. (2013) Molecular dynamics simulations accelerated by GPU for biological macromolecules with a non-Ewald scheme for electrostatic interactions. *Journal of Chemical Theory and Computation* 9(12): 5599–5609.

Meng Q, Luitjens J and Berzins M (2010) Dynamic task scheduling for the Uintah framework. In: *IEEE workshop on many-task computing on grids and supercomputers (MTAGS)*, New Orleans, LA, 15 November, pp.1–10. IEEE.

Metropolis N, Rosenbluth AW, Rosenbluth MN, et al. (1953) Equation of state calculations by fast computing machines. *The Journal of Chemical Physics* 21(6): 1087–1092. DOI: 10.1063/1.1699114.

Mura ME and Knowles PJ (1996) Improved radial grids for quadrature in molecular density-functional calculations. *The Journal of Chemical Physics* 104(24): 9848–9858.

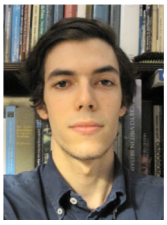
Nitsche MA, Ferreria M, Mocskos EE, et al. (2014) GPU accelerated implementation of density functional theory for hybrid QM/MM simulations. *Journal of Chemical Theory and Computation* 10(3): 959–967.

Perdew JP, Burke K and Ernzerhof M (1996) Generalized gradient approximation made simple. *Physical Review Letters* 77(18): 3865–3868.

Pratas F, Sousa L, Dieterich JM, et al. (2012) Computation of induced dipoles in molecular mechanics simulations using

graphics processors. *Journal of Chemical Information and Modeling* 52(5): 1159–1166.
Stone JE, Phillips JC, Freddolino PL, et al. (2007) Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry* 28(16): 2618–2640.
Taniça L, Ilic A, Tomás P, et al. (2014) Schedmon: A performance and energy monitoring tool for modern multi-cores. In: *Euro-Par 2014: Parallel Processing Workshops, Lecture Notes in Computer Science*, volume 8806. Cham, Switzerland: Springer International Publishing, pp.230–241.
Thakur R and Gropp WD (2003) Improving the performance of collective operations in MPICH. In: Dongarra J, Laforenza D and Orlando S (eds) *Recent advances in parallel virtual machine and message passing interface: 10th European PVM/MPI User's Group Meeting*, Venice, Italy, 29 September–2 October, pp. 257–267. DOI: 10.1007/978-3-540-39924-7_38.
Truong TN and Stefanovich EV (1996) Development of a perturbative approach for Monte Carlo simulations using a hybrid ab initio QM/MM method. *Chemical Physics Letters* 256(3): 348–352.
Uejima Y, Terashima T and Maezono R (2011) Acceleration of a QM/MM-QMC simulation using GPU. *Journal of Computational Chemistry* 32(10): 2264–2272.
Weigend F and Ahlrichs R (2005) Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for H to Rn: Design and assessment of accuracy. *Physical Chemistry Chemical Physics* 7(18): 3297–3305.
Werner HJ, Knowles PJ, Knizia G, et al. (2012) Molpro: A general-purpose quantum chemistry program package. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 2(2): 242–253.
Zheng G, Meneses E, Bhatele A, et al. (2010) Hierarchical load balancing for charm++ applications on large supercomputers. In: *39th International Conference on Parallel Processing Workshops*, San Diego, CA, 13–16 September, pp. 436–444. DOI: 10.1109/ICPPW.2010.65.

Author Biographies



Sebastião Miranda received his MSc in Electrical and Computer Engineering in 2014 from Instituto Superior Técnico, University of Lisbon, Portugal. In 2013, he joined INESC-ID as a MSc student of the SiPS group, where he started his research on exploiting highly parallel heterogeneous platforms to accelerate compute-intensive scientific applications. He is also interested in machine learning, artificial intelligence and embedded systems.

Jonas Feldt received his MSc in Chemistry from the University of Göttingen, Germany, in 2014. He is currently a PhD student at the Computational Chemistry and Biochemistry group. His research focus lies on simulation methods for systems in the condensed phase.

Frederico Pratas received his PhD degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade de Lisboa, Lisbon, Portugal, in 2012. Until 2013, he was also a researcher at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), Lisbon, Portugal, with whom he still has some ongoing collaborations. He has recently joined the Imagination Technologies' MIPS group at KL, UK, as a Hardware Design Engineer, where he collaborates on randomized testing for system level HW verification of future processors. His research interests include computer architectures and microarchitectures, high-performance computing, HW design and verification and reconfigurable computing.

Ricardo A. Mata earned his PhD in theoretical chemistry in 2007 from the University of Stuttgart, under the supervision of Prof Hans-Joachim Werner. He then joined the University of Lisbon in the Mathematical Physics Group, where he worked on the development of incremental approaches to the study of electronic excitation spectra in solution. In 2009 he joined the University of Göttingen as a Free-Floater junior research group leader. In late 2015 he became a full-time professor in the Faculty of Chemistry. His current research focuses on the study of reactivity and weak interactions in biomolecular systems, through the use of local and incremental correlation schemes.

Nuno Roma received his PhD degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade de Lisboa, Lisbon, Portugal in 2008. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering of IST and a Senior Researcher of the Signal

Processing Systems Group (SiPS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include computer architectures, specialized and dedicated structures for digital signal processing (including image and video coding and biological sequences processing), parallel processing and high-performance computing systems. He has contributed more than 80 papers to journals and international conferences. Dr Roma is a Senior Member of the IEEE Circuits and Systems Society and a member of ACM.

Pedro Tomás received a five-year engineering degree, MSc and PhD degrees in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal, in 2003, 2006 and 2009, respectively. Currently he is an Assistant Professor at the Department of Electrical and Computer Engineering of IST and a Researcher at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include computer architectures and micro-architectures, high-performance computing and signal processing for biomedical applications. He is a member of the IEEE Computer Society and has contributed more than 45 papers to international peer-reviewed journals and conferences.