Computational Physics on Graphics Processing Units

Ari Harju[1,2], Topi Siro[1,2], Filippo Federici Canova[3], Samuli Hakala[1], and Teemu Rantalaiho[2,4]

[1] COMP Centre of Excellence, Department of Applied Physics, Aalto University School of Science, Helsinki, Finland
[2] Helsinki Institute of Physics, Helsinki, Finland
[3] Department of Physics, Tampere University of Technology, Tampere, Finland
[4] Department of Physics, University of Helsinki, Helsinki, Finland

Abstract. The use of graphics processing units for scientific computations is an emerging strategy that can significantly speed up various algorithms. In this review, we discuss advances made in the field of computational physics, focusing on classical molecular dynamics and quantum simulations for electronic structure calculations using the density functional theory, wave function techniques and quantum field theory.

Keywords: graphics processing units, computational physics.

1 Introduction

The graphics processing unit (GPU) has been an essential part of personal computers for decades. Their role became much more important in the 90s when the era of 3D graphics in gaming started. One of the hallmarks of this is the violent first-person shooting game DOOM by the id Software company, released in 1993. Wandering around the halls of slaughter, it was hard to imagine these games leading to any respectable science. However, twenty years after the release of DOOM, the gaming industry of today is enormous, and the continuous need for more realistic visualizations has led to a situation where modern GPUs have tremendous computational power. In terms of theoretical peak performance, they have far surpassed the central processing units (CPU).

The games started to have real 3D models and hardware acceleration in the mid 90s, but an important turning point for the scientific use of GPUs for computing was around the first years of this millennium [1], when the widespread programmability of GPUs was introduced. Combined with the continued increase in computational power as shown in Fig. 1, the GPUs are nowadays a serious platform for general purpose computing. Also, the memory bandwidth in GPUs is very impressive. The three main vendors for GPUs, Intel, NVIDIA, and ATI/AMD, are all actively developing computing on GPUs. At the moment, none of the technologies listed above dominate the field, but NVIDIA with its CUDA programming environment is perhaps the current market leader.

P. Manninen and P. Öster (Eds.): PARA 2012, LNCS 7782, pp. 3–26, 2013.

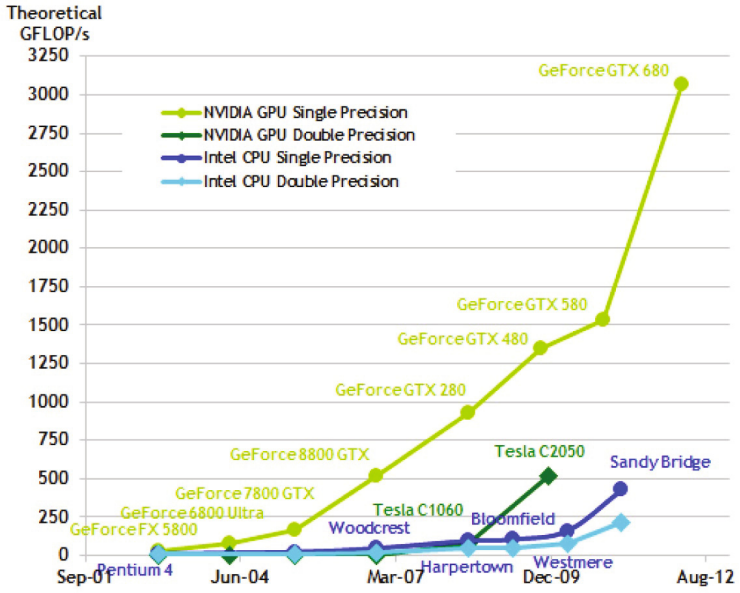

Fig. 1. Floating point operations (FLOPS) per second for GPUs and CPUs from NVIDIA and Intel Corporations, figure taken from [2]. The processing power of the currently best GPU hardware by the AMD Corporation is comparable to NVIDIA at around 2600 GFLOPS/s.

1.1 The GPU as a Computational Platform

At this point, we have hopefully convinced the reader that GPUs feature a powerful architecture also for general computing, but what makes GPUs different from the current multi-core CPUs? To understand this, we can start with traditional graphics processing, where hardware vendors have tried to maximize the speed at which the pixels on the screen are calculated. These pixels are independent primitives that can be processed in parallel, and the number of pixels on computer displays has increased over the years from the original DOOM resolution of 320×200, corresponding to 64000 pixels, to millions. The most efficient way to process these primitives is to have a very large number of arithmetic logical units (ALUs) that are able to perform a high number of operations for each video frame. The processing is very data-parallel, and one can view this as performing the same arithmetic operation in parallel for each primitive. Furthermore, as the operation is the same for each primitive, there is no need for very sophisticated flow control in the GPU and more transistors can be used for arithmetics, resulting in an enormously efficient hardware for performing parallel computing that can be classified as "single instruction, multiple data" (SIMD).

Now, for general computing on the GPU, the primitives are no longer the pixels on the video stream, but can range from matrix elements in linear algebra to physics related cases where the primitives can be particle coordinates

in classical molecular dynamics or quantum field values. Traditional graphics processing teaches us that the computation would be efficient when we have a situation where the same calculation needs to be performed for each member of a large data set. It is clear that not all problems or algorithms have this structure, but there are luckily many cases where this applies, and the list of successful examples is long.

However, there are also limitations on GPU computing. First of all, when porting a CPU solution of a given problem to the GPU, one might need to change the algorithm to suit the SIMD approach. Secondly, the communication from the host part of the computer to the GPU part is limited by the speed of the PCIe bus coupling the GPU and the host. In practice, this means that one needs to perform a serious amount of computing on the GPU between the data transfers before the GPU can actually speed up the overall computation. Of course, there are also cases where the computation as a whole is done on GPU, but these cases suffer from the somewhat slower serial processing speed of the GPU.

Additional challenges in GPU computing include the often substantial programming effort to get a working and optimized code. While writing efficient GPU code has become easier due to libraries and programmer friendly hardware features, it still requires some specialized thinking. For example, the programmer has to be familiar with the different kinds of memory on the GPU to know how and when to use them. Further, things like occupancy of the multiprocessors (essentially, how full the GPU is) and memory access patterns of the threads are something one has to consider to reach optimal performance. Fortunately, each generation of GPUs has alleviated the trouble of utilizing their full potential. For example, a badly aligned memory access in the first CUDA capable GPUs from NVIDIA could cripple the performance by drastically reducing the memory bandwidth, while in the Fermi generation GPUs the requirements for memory access coalescing are much more forgiving.

2 Molecular Dynamics

Particle dynamics simulation, often simply called Molecular dynamics (MD), refers to the type of simulation where the behaviour of a complex system is calculated by integrating the equation of motion of its components within a given model, and its goal is to observe how some ensemble-averaged properties of the system originate from the detailed configuration of its constituent particles (Fig. 2).

In its classical formulation, the dynamics of a system of particles is described by their Newtonian equations:

$$m_i \frac{d^2 \boldsymbol{x}_i}{dt^2} = \sum_j \boldsymbol{F}_{ij} \tag{1}$$

where m_i is the particle's mass, $\boldsymbol{x}_i$ its position, and $\boldsymbol{F}_{ij}$ is the interaction between the i-th and j-th particles as provided by the model chosen for the system under

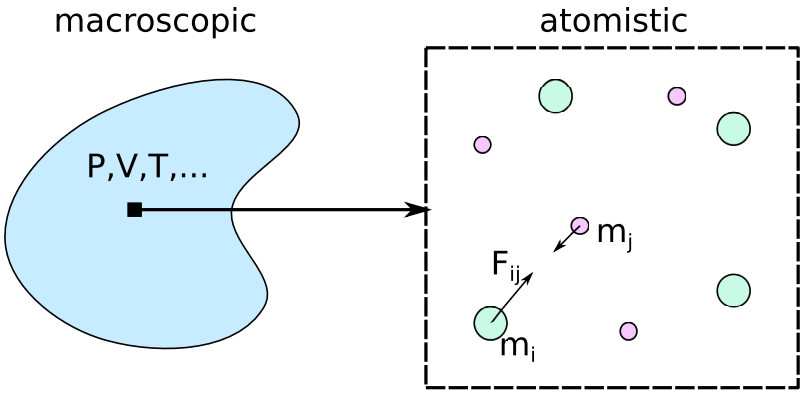

Fig. 2. Schematic presentation of the atomistic model of a macroscopic system

study. These second order differential equations are then discretised in the time domain, and integrated step by step until a convergence criterion is satisfied.

The principles behind MD are so simple and general that since its first appearance in the 70s, it has been applied to a wide range of systems, at very different scales. For example, MD is the dominant theoretical tool of investigation in the field of biophysics, where structural changes in proteins [3,4,5,6] and lipid bilayers [7,8] interacting with drugs can be studied, ultimately providing a better understanding of drug delivery mechanisms.

At larger scales, one of the most famous examples is known as the *Millenium Simulation*, where the dynamics of the mass distribution of the universe at the age of 380000 years was simulated up to the present day [9], giving an estimate of the age of cosmic objects such as galaxies, black holes and quasars, greatly improving our understanding of cosmological models and providing a theoretical comparison to satellite measurements.

Despite the simplicity and elegance of its formulation, MD is not a computationally easy task and often requires special infrastructure. The main issue is usually the evaluation of all the interactions $\boldsymbol{F}_{ij}$, which is the most time consuming procedure of any MD calculation for large systems. Moreover, the processes under study might have long characteristic time scales, requiring longer simulation time and larger data storage; classical dynamics is chaotic, i.e. the outcome is affected by the initial conditions, and since these are in principle unknown and chosen at random, some particular processes of interest might not occur just because of the specific choice, and the simulation should be repeated several times. For these reasons, it is important to optimise the evaluation of the forces as much as possible.

An early attempt to implement MD on the GPU was proposed in 2004 [10] and showed promising performance; at that time, general purpose GPU computing was not yet a well established framework and the N-body problem had to be formulated as a rendering task: a *shader* program computed each pair interaction $\boldsymbol{F}_{ij}$ and stored them as the pixel color values (RBG) in an $N \times N$ texture. Then,

another shader would simply sum these values row-wise to obtain the total force on each particle and finally integrate their velocities and positions. The method is called all-pairs calculation, and as the name might suggest, it is quite expensive as it requires $\mathcal{O}(N^2)$ force evaluations. The proposed implementation was in no way optimal since the measured performance was about a tenth of the nominal value of the device, and it immediately revealed one of the main issues of the architecture that still persists nowadays: GPUs can have a processing power exceeding the teraflop, but, at the same time, they are extremely slow at handling the data to process since a memory read can require hundreds of clock cycles. The reason for the bad performance was in fact the large amount of memory read instructions compared to the amount of computation effectively performed on the fetched data, but despite this limitation, the code still outperformed a CPU by a factor of 8 because every interaction was computed concurrently. A wide overview of optimisation strategies to get around the memory latency issues can be found in Ref. [11], while, for the less eager to get their hands dirty, a review of available MD software packages is included in Ref. [12].

In the current GPU programming model, the computation is distributed in different threads, grouped together as blocks in a grid fashion, and they are allowed to share data and synchronise throughout the same block; the hardware also offers one or two levels of cache to enhance data reuse, thus reducing the amount of memory accesses, without harassing the programmer with manual prefetching. A more recent implementation of the all-pair calculation [13] exploiting the full power of the GPU can achieve a performance close to the nominal values, comparable to several CPU nodes.

The present and more mature GPGPU framework allows for more elaborate kernels to fit in the device, enabling the implementation of computational tricks developed during the early days of MD [14] that make it possible to integrate N-body dynamics accurately with much better scaling than $\mathcal{O}(N^2)$. For example, in many cases the inter-particle forces are short range, and it would be unnecessary to evaluate every single interaction $\boldsymbol{F}_{ij}$ since quite many of them would be close to zero and just be neglected. It is good practice to build lists of neighbours for each particle in order to speed up the calculation of forces: this also takes an $\mathcal{O}(N^2)$ operation, although the list is usually only recalculated every 100-1000 timesteps, depending on the average mobility of the particles. The optimal way to build neighbour lists is to divide the simulation box in voxels and search for a partcle's neighbours only within the adjacent voxels (Fig. 3a), as this procedure requires only $\mathcal{O}(N)$ instructions. Performance can be further improved by sorting particles depending on the index of the voxel they belong, making neighbouring particles in space, to a degree, close in memory, thus increasing coalescence and cache hit rate on GPU systems; such a task can be done with radix count sort [15,16,17] in $\mathcal{O}(N)$ with excellent performance, and it was shown to be the winning strategy [18].

Unfortunately, most often the inter-particle interactions are not exclusively short range and can be significant even at larger distances (electrostatic and gravitational forces). Therefore, introducing an interaction cut-off leads to the

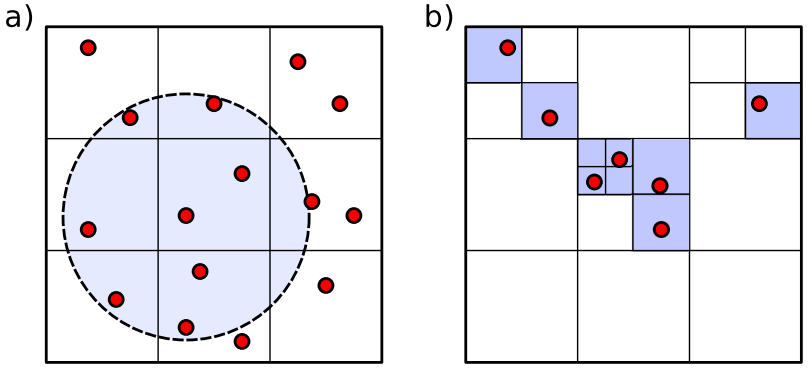

Fig. 3. Illustration of different space partition methods. In dense systems (a) a regular grid is preferred, and neighbouring particles can be searched in only a few adjacent voxels. Sparse systems (b) are better described by hierarchical trees, excluding empty regions from the computation.

wrong dynamics. For dense systems, such as bulk crystals or liquids, the electrostatic interaction length largely exceeds the size of the simulation space, and in principle one would have to include the contributions from several periodic images of the system, although their sum is not always convergent. The preferred approach consists of calculating the electrostatic potential $V(\boldsymbol{r})$ generated by the distribution of point charges $\rho(\boldsymbol{r})$ from Poisson's equation:

$$\nabla^2 V(\boldsymbol{r}) = \rho(\boldsymbol{r}) \tag{2}$$

The electrostatic potential can be calculated by discretising the charge distribution on a grid, and solving Eq. 2 with a fast Fourier transform (FFT), which has $\mathcal{O}(MlogM)$ complexity (where M is the amount of grid points): this approach is called particle-mesh Ewald (PME). Despite being heavily non-local, much work has been done to improve the FFT algorithm and make it cache efficient [19,20,21,22,23], so it is possible to achieve a 20-fold speed up over the standard CPU FFTW or a 5-fold speedup when compared to a highly optimised MKL implementation. The more recent multilevel summation method (MSM) [24] uses nested interpolations of progressive smoothing of the electrostatic potential on lattices with different resolutions, offering a good approximation of the electrostatic $\mathcal{O}(N^2)$ problem in just $\mathcal{O}(N)$ operations. The advantage of this approach is the simplicity of its parallel implementation, since it requires less memory communication among the nodes, which leads to a better scaling than the FFT calculation in PME. The GPU implementation of this method gave a 25-fold speedup over the single CPU [25]. Equation 2 can also be translated into a linear algebra problem using finite differences, and solved iteratively on multigrids [26,27] in theoretically $\mathcal{O}(M)$ operations. Even though the method initially requires several iterations to converge, the solution does not change much in one MD step and can be used as a starting point in the following step, which in turn will take much fewer iterations.

On the other hand, for sparse systems such as stars in cosmological simulations, decomposing the computational domain in regular boxes can be quite harmful because most of the voxels will be empty and some computing power and memory is wasted there. The optimal way to deal with such a situation is to subdivide the space hierarchically with an octree [28] (Fig. 3b), where only the subregions containing particles are further divided and stored. Finding neighbouring particles can be done via a traversal of the tree in $\mathcal{O}(NlogN)$ operations. Octrees are conventionally implemented on the CPU as dynamical data structures where every node contains reference pointers to its parent and children, and possibly information regarding its position and content. This method is not particularly GPU friendly since the data is scattered in memory as well as in the simulation space. In GPU implementations, the non-empty nodes are stored as consecutive elements in an array or texture, and they include the indices of the children nodes [29]. They were proved to give a good acceleration in solving the N-body problem [13,30,31]. Long range interactions are then calculated explicitly for the near neighbours, while the fast multipole method (FMM) [32,33] can be used to evaluate contributions from distant particles. The advantage of representing the system with an octree becomes now more evident: there exists a tree node containing a collection of distant particles, which can be treated as a single multipole leading to an overall complexity $\mathcal{O}(N)$. Although the mathematics required by FMM is quite intensive to evaluate, the algorithms involved have been developed and extensively optimised for the GPU architecture [34,35,36], achieving excellent parallel performance even on large clusters [37].

In all the examples shown here, the GPU implementation of the method outperformed its CPU counterpart: in many cases the speedup is only 4-5 fold when compared to a highly optimised CPU code, which seems, in a way, a discouraging result, because implementing an efficient GPU algorithm is quite a difficult task, requiring knowledge of the target hardware, and the programming model is not as intuitive as for a regular CPU. To a degree, the very same is true for CPU programming, where taking into account cache size, network layout, and details of shared/distributed memory of the target machine when designing a code leads to higher performance. These implementation difficulties could be eased by developing better compilers, that check how memory is effectively accessed and provide higher levels of GPU optimisation on older CPU codes automatically, hiding the complexity of the hardware specification from the programmer. In some cases, up to 100 fold speedups were measured, suggesting that the GPU is far superior. These cases might be unrealistic since the nominal peak performance of a GPU is around 5 times bigger than that of a CPU. Therefore, it is possible that the benchmark is done against a poorly optimised CPU code, and the speedup is exaggerated. On the other hand, GPUs were also proven to give good scaling in MPI parallel calculations, as shown in Refs. [31] and [37]. In particular, the AMBER code was extensively benchmarked in Ref. [38], and it was shown how just a few GPUs (and even just one) can outperform the same code running on 1024 CPU cores: the weight of the communication between nodes exceeds the benefit of having additional CPU cores, while the few GPUs do not

suffer from this latency and can deliver better performance, although the size of the computable system becomes limited by the available GPU memory. It has to be noted how GPU solutions, even offering a modest 4-5 fold speedup, do so at a lower hardware and running cost than the equivalent in CPUs, and this will surely make them more appealing in the future. From the wide range of examples in computational physics, it is clear that the GPU architecture is well suited for a defined group of problems, such as certain procedures required in MD, while it fails for others. This point is quite similar to the everlasting dispute between raytracing and raster graphics: the former can explicitly calculate photorealistic images in complex scenes, taking its time (CPU), while the latter resorts to every trick in the book to get a visually "alright" result as fast as possible (GPU). It would be best to use both methods to calculate what they are good for, and this sets a clear view of the future hardware required for scientific computing, where both simple vector-like processors and larger CPU cores could access the same memory resources, avoiding data transfer.

3 Density-Functional Theory

Density functional theory (DFT) is a popular method for *ab-initio* electronic structure calculations in material physics and quantum chemistry. In the most commonly used DFT formulation by Kohn and Sham [39], the problem of N interacting electrons is mapped to one with N non-interacting electrons moving in an effective potential so that the total electron density is the same as in the original many-body case [40]. To be more specific, the single-particle Kohn-Sham orbitals $\psi_n(\mathbf{r})$ are solutions to the equation

$$H\psi_n(\mathbf{r}) = \epsilon_n \psi_n(\mathbf{r}), \tag{3}$$

where the effective Hamiltonian in atomic units is $H = -\frac{1}{2}\nabla^2 + v_H(\mathbf{r}) + v_{ext}(\mathbf{r}) + v_{xc}(\mathbf{r})$. The three last terms in the Hamiltonian define the effective potential, consisting of the Hartree potential v_H defined by the Poisson equation $\nabla^2 v_H(\mathbf{r}) = -4\pi\rho(\mathbf{r})$, the external ionic potential v_{ext}, and the exchange-correlation potential v_{xc} that contains all the complicated many-body physics the Kohn-Sham formulation partially hides. In practice, the v_{xc} part needs to be approximated. The electronic charge density $\rho(\mathbf{r})$ is determined by the Kohn-Sham orbitals as $\rho(\mathbf{r}) = \sum_i f_i |\psi_i(\mathbf{r})|^2$, where the f_i:s are the orbital occupation numbers.

There are several numerical approaches and approximations for solving the Kohn-Sham equations. They relate usually to the discretization of the equations and the treatment of the core electrons (pseudo-potential and all electron methods). The most common discretization methods in solid state physics are plane waves, localized orbitals, real space grids and finite elements. Normally, an iterative procedure called self-consistent field (SCF) calculation is used to find the solution to the eigenproblem starting from an initial guess for the charge density [41].

Porting an existing DFT code to GPUs generally includes profiling or discovering with some other method the computationally most expensive parts of the

SCF loop and reimplementing them with GPUs. Depending on the discretization methods, the known numerical bottlenecks are vector operations, matrix products, Fast Fourier Transforms (FFTs) and stencil operations. There are GPU versions of many of the standard computational libraries (like CUBLAS for BLAS and CUFFT for FFTW). However, porting a DFT application is not as simple as replacing the calls to standard libraries with GPU equivalents since the resulting intermediate data usually gets reused by non standard and less computationally intensive routines. Attaining high performance on a GPU and minimizing the slow transfers between the host and the device requires writing custom kernels and also porting a lot of the non-intensive routines to the GPU.

Gaussian basis functions are a popular choice in quantum chemistry to investigate electronic structures and their properties. They are used in both DFT and Hartree-Fock calculations. The known computational bottlenecks are the evaluation of the two-electron repulsion integrals (ERIs) and the calculation of the exchange-correlation potential. Yasuda was the first to use GPUs in the calculation of the exchange-correlation term [42] and in the evaluation of the Coulomb potential [43]. The most complete work in this area was done by Ufimtsev *et al.*. They have used GPUs in ERIs [44,45,46], in complete SCF calculations [47] and in energy gradients [48]. Compared to the mature GAMESS quantum chemistry package running on CPUs, they were able to achieve speedups of more than 100 using mixed precision arithmetic in HF SCF calculations. Asadchev *et al.*. have also done an ERI implementation on GPUs using the uncontracted Rys quadrature algorithm [49].

The first complete DFT code on GPUs for solid state physics was presented by Genovese *et al.*. [50]. They used double precision arithmetic and a Daubechies wavelet based code called BIGDFT [51]. The basic 3D operations for a wavelet based code are based on convolutions. They achieved speedups of factor 20 for some of these operations on a GPU, and a factor of 6 for the whole hybrid code using NVIDIA Tesla S1070 cards. These results were obtained on a 12-node hybrid machine.

For solid state physics, plane wave basis sets are the most common choice. The computational schemes rely heavily on linear algebra operations and fast Fourier transforms. The Vienna ab initio Simulation Package (VASP) [52] is a popular code combining plane waves with the projector augmented wave method. The most time consuming part of optimizing the wave functions given the trial wave functions and related routines have been ported to GPUs. Speedups of a factor between 3 and 8 for the blocked Davinson scheme [53] and for the RMM-DIIS algorithm [54] were achieved in real-world examples with Fermi C2070 cards. Parallel scalability with 16 GPUs was similar to 16 CPUs. Additionally, Hutchinson *et al.* have done an implementation of exact-exchange calculations on GPUs for VASP [55].

Quantum ESPRESSO [56] is a electronic structure code based on plane wave basis sets and pseudo-potentials (PP). For the GPU version [57], the most computationally expensive parts of the SCF cycle were gradually transferred to run on GPUs. FFTs were accelerated by CUFFT, LAPACK by MAGMA and other

routines were replaced by CUDA kernels. GEMM operations were replaced by the parallel hybrid phiGEMM [58] library. For single node test systems, running with NVIDIA Tesla C2050, speedups between 5.5 and 7.8 were achieved and for a 32 node parallel system speedups between 2.5 and 3.5 were observed. Wand et al. [59] and Jia *et al.*. [60] have done an implementation for GPU clusters of a plane wave pseudo-potential code called PEtot. They were able to achieve speedups of 13 to 22 and parallel scalability up to 256 CPU-GPU computing units.

GPAW [61] is a density-functional theory (DFT) electronic structure program package based on the real space grid based projector augmented wave method. We have used GPUs to speed up most of the computationally intensive parts of the code: solving the Poisson equation, iterative refinement of the eigenvectors, subspace diagonalization and orthonormalization of the wave functions. Overall, we have achieved speedups of up to 15 on large systems and a good parallel scalability with up to 200 GPUs using NVIDIA Tesla M2070 cards [62].

Octopus [63,64] is a DFT code with an emphasis on the time-dependent density-functional theory (TDDFT) using real space grids and pseudo-potentials. Their GPU version uses blocks of Kohn-Sham orbitals as basic data units. Octopus uses GPUs to accelerate both time-propagation and ground state calculations. Finally, we would like to mention the linear response Tamm-Dancoff TDDFT implementation [65] done for the GPU-based TeraChem code.

4 Quantum Field Theory

Quantum field theories are currently our best models for fundamental interactions of the natural world (for a brief introduction to quantum field theories – or QFTs – see for example [66] or [67] and references therein). Common computational techniques include perturbation theory, which works well in quantum field theories as long as the couplings are small enough to be considered as perturbations to the free theory. Therefore, perturbation theory is the primary tool used in pure QED, weak nuclear force and high momentum-transfer QCD phenomena, but it breaks up when the coupling constant of the theory (the measure of the interaction strength) becomes large, such as in low-energy QCD.

Formulating the quantum field theory on a space-time lattice provides an opportunity to study the model non-perturbatively and use computer simulations to get results for a wide range of phenomena – it enables, for example, one to compute the hadronic spectrum of QCD (see [68] and references therein) from first principles and provides solutions for many vital gaps left by the perturbation theory, such as structure functions of composite particles [69], form-factors [70] and decay-constants [71]. It also enables one to study and test models for new physics, such as technicolor theories [72] and quantum field theories at finite temperature [73], [74] or [75]. For an introduction to Lattice QFT, see for example [76], [77] or [78].

Simulating quantum field theories using GPUs is not a completely new idea and early adopters even used OpenGL (graphics processing library) to program

the GPUs to solve lattice QCD [79]. The early GPGPU programmers needed to set up a program that draws two triangles that fill the output texture of desired size by running a "shader program" that does the actual computation for each output pixel. In this program, the input data could be then accessed by fetching pixels' input texture(s) using the texture units of the GPU. In lattice QFT, where one typically needs to fetch the nearest neighbor lattice site values, this actually results in good performance as the texture caches and layouts of the GPUs have been optimized for local access patterns for filtering purposes.

4.1 Solving QFTs Numerically

The idea behind lattice QFT is based on the discretization of the path integral solution to expectation values of time-ordered operators in quantum field theories. First, one divides spacetime into discrete boxes, called the lattice, and places the fields onto the lattice sites and onto the links between the sites, as shown in Fig. 4. Then, one can simulate nature by creating a set of multiple field configurations, called an *ensemble*, and calculate the values of physical observables by computing ensemble averages over these states.

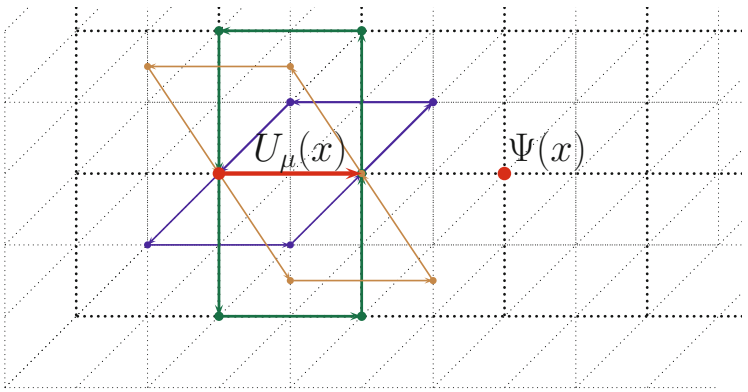

Fig. 4. The matter fields $\Psi(x)$ live on lattice sites, whereas the gauge fields $U_\mu(x)$ live on the links connecting the sites. Also depicted are the staples connecting to a single link variable that are needed in the computation of the gauge field forces.

The set of states is normally produced with the help of a Markov chain and in the most widely studied QFT, the lattice QCD, the chain is produced by combining a *molecular dynamics* algorithm together with a *Metropolis* acceptance test. Therefore, the typical computational tasks in lattice QFTs are:

1. Refresh generalized momentum variables from a heat bath (Gaussian distribution) once per *trajectory*.
2. Compute generalized forces for fields for each step

3. Integrate classical equations of motion for the fields at each step [1]
4. Perform a Metropolis acceptance test at the end of the trajectory in order to achieve the correct limiting distribution.

In order to reach satisfying statistics, normally thousands of these trajectories need to be generated and each trajectory is typically composed of 10 to 100 steps. The force calculation normally involves a matrix inversion, where the matrix indices run over the entire lattice and it is therefore the heaviest part of the computation. The matrix arises in simulations with dynamical fermions (normal propagating matter particles) and the simplest form for the fermion matrix is[2]

$$A_{x,y} = [Q^\dagger Q]_{x,y} \qquad \text{where}$$

$$Q_{x,y} = \delta_{x,y} - \kappa \sum_{\mu=\pm 1}^{\pm 4} \delta_{y+\hat{\mu},x}(1+\gamma_\mu)U_\mu(x). \tag{4}$$

Here, κ is a constant related to the mass(es) of the quark(s), $\delta_{x,y}$ is the *Kronecker delta function* (unit matrix elements), the sum goes over the spacetime dimensions μ, γ_μ are 4-by-4 constant matrices and $U_\mu(x)$ are the link variable matrices that carry the force (gluons for example) from one lattice site to the neighbouring one. In normal QCD they are 3-by-3 complex matrices.

The matrix A in the equation $Ar = z$, where one solves for the vector r with a given z, is an almost diagonal sparse matrix with a *predefined sparsity pattern*. This fact makes lattice QCD ideal for parallelization, as the amount work done by each site is constant. The actual algorithm used in the matrix inversion is normally some variant of the conjugate gradient algorithm, and therefore one needs fast code to handle the multiplication of a fermion vector by the fermion matrix.

This procedure is the generation of the lattice configurations which form the ensemble. Once the set of configurations $\{U_i\}, i \in [1, N]$ has been generated with the statistical weight $e^{-S[U_i]}$, where $S[U_i]$ is the *Euclidean action* (action in imaginary time formulation), the expectation value of an operator $F[U]$ can be computed simply as

$$\langle F[U] \rangle \approx \frac{1}{N} \sum_{i=1}^{N} F[U_i], \tag{5}$$

4.2 Existing GPU Solutions to Lattice QFTs

As lattice QFTs are normally easily parallelizable, they fit well into the GPU programming paradigm, which can be characterized as parallel throughput computation. The conjugate gradient methods perform many fermion matrix vector multiplications whose arithmetic intensity (ratio of floating point operations done per byte of memory fetched) is quite low, making memory bandwidth the

[1] The integration is *not* done with respect to normal time variable, but through the Markov chain index-"time".

[2] There are multiple different algorithms for simulating fermions, here we present the simplest one for illustrative purposes.

normal bottleneck within a single processor. Parallelization between processors is done by standard MPI domain decomposition techniques. The conventional wisdom that this helps due to higher local volume to communication surface area ratio is actually flawed, as typically the GPU can handle a larger volume in the same amount of time, hence requiring the MPI-implementation to also take care of a larger surface area in the same time as with a CPU. In our experience, GPU adoption is still in some sense in its infancy, as the network implementation seems to quickly become the bottleneck in the computation and the MPI implementations of running systems seem to have been tailored to meet the needs of the CPUs of the system. Another aspect of this is that normally the GPUs are coupled with highly powered CPUs in order to cater for the situation where the users use the GPUs in just a small part of the program and need a lot of sequential performance in order to try to keep the serial part of the program up with the parallel part. The GPU also needs a lot of concurrent threads (in the order of thousands) to be filled completely with work and therefore good performance is only achievable with relatively large local lattice sizes.

Typical implementations assign one GPU thread per site, which makes parallelization easy and gives the compiler quite a lot of room to find instruction level parallelism, but in our experience this can result in a relatively high register pressure: the quantum fields living on the sites have many indices (normally color and Dirac indices) and are therefore vectors or matrices with up to 12 complex numbers per field per site in the case of quark fields in normal QCD. Higher parallelization can be achieved by taking advantage of the vector-like parallelism inside a single lattice site, but this may be challenging to implement in those loops where the threads within a site have to collaborate to produce a result, especially because GPUs impose restrictions on the memory layout of the fields (consecutive threads have to read consecutive memory locations in order to reach optimal performance [2]). In a recent paper [80], the authors solve the *gauge fixing* problem by using overrelaxation techniques and they report an increase in performance by using multiple threads per site, although in this case the register pressure problem is even more pronounced and the effects of register spilling to the L1 cache were not studied.

The lattice QCD community has a history of taking advantage of computing solutions outside the mainstream: the QCDSP [81] computer was a custom machine that used digital signal processors to solve QCD with an order of one teraflop of performance. QCDOC [82] used a custom IMB powerPC-based ASIC and a multidimensional torus network, which later on evolved into the first version of the Blue Gene supercomputers [83]. The APE collaboration has a long history of custom solutions for lattice QCD and is building custom network solutions for lattice QCD [84]. For example, QCDPAX [85] was a very early parallel architecture used to study Lattice QCD without dynamical fermions.

Currently, there are various groups using GPUs to do lattice QFT simulations. The first results using GPUs were produced as early as 2006 in a study that determined the transition temperature of QCD [86]. Standardization efforts for high precision Lattice QCD libraries are underway and the QUDA library [87] scales to hundreds of GPUs by using a local Schwarz preconditioning technique,

effectively eliminating all the GPU-based MPI communications for a significant portion of the calculation. They employ various optimization techniques, such as *mixed-precision* solvers, where parts of the inversion process of the fermion matrix is done at lower precision of floating point arithmetic and using reduced representations of the SU3 matrices. Scaling to multiple GPUs can also be improved algorithmically: already a simple (almost standard) *clover improvement* [88] term in the fermion action leads to better locality and of course improves the action of the model as well, taking the lattice formulation closer to the continuum limit. Domain decomposition and taking advantage of restricted additive Schwarz (RAS) preconditioning using GPUs was already studied in 2010 in [89], where the authors get the best performance on a 32^4 lattice with vanishing overlap between the preconditioning domains and three complete RAS iterations each containing just five iterations to solve the local system of 4×32^3 sites. It should be noted though that the hardware they used is already old, so optimal parameters with up-to-date components could slightly differ.

Very soon after starting to work with GPUs on lattice QFTs, one notices the effects of Amdahl's law which just points out the fact that there is an upper bound for the whole program performance improvement related to optimizing just a portion of the program. It is quite possible that the fermion matrix inversion takes up 90% of the total computing time, but making this portion of the code run 10 times faster reveals something odd: now we are spending half of our time computing forces and doing auxiliary computations and if we optimize this portion of the code as well, we improve our performance by a factor of almost two again – therefore optimizing only the matrix inversion gives us a mere fivefold performance improvement instead of the promised order of magnitude improvement. Authors of [90] implemented practically the entire HMC trajectory on the GPU to fight Amdahl's law and recent work [91] on the QDP++ library implements *Just-in-Time* compilation to create GPU kernels on the fly to accommodate any non-performance critical operation over the entire lattice.

Work outside of standard Lattice QCD using GPUs includes the implementation of the Neuberger-Dirac overlap operator [92], which provides *chiral symmetry* at the expense of a non-local action. Another group uses the Arnoldi algorithm on a multi-GPU cluster to solve the overlap operator [93] and shows scaling up to 32 GPUs. Quenched SU2 [94] and later quenched SU2, SU3 and generic SU(N_c) simulations using GPUs are described in [95] and even compact U(1) Polyakov loops using GPUs are studied in [96]. Scalar field theory – the so-called $\lambda\phi^4$ model – using AMD GPUs is studied in [97]. The TWQCD collaboration has also implemented almost the entire HMC trajectory computation with dynamical Optimal Domain Wall Fermions, which improve the chiral symmetry of the action [98].

While most of the groups use exclusively NVIDIA's CUDA-implementation [2], which offers good reliability, flexibility and stability, there are also some groups using the OpenCL standard [99]. A recent study [100] showed better performance on AMD GPUs than on NVIDIA ones using OpenCL, although it should be noted that the NVIDIA GPUs were consumer variants with reduced

double precision throughput and that optimization was done for AMD GPUs. The authors of [90] have implemented both CUDA and OpenCL versions of their staggered fermions code and they report a slightly higher performance for CUDA and for NVIDIA cards.

4.3 QFT Summary

All in all, lattice QFT using GPUs is turning from being a promising technology to a very viable alternative to traditional CPU-based computing. When reaching for the very best strong scaling performance – meaning best performance for small lattices – single threaded performance does matter if we assume that the rest of the system scales to remove other bottlenecks (communication, memory bandwith.) In these cases, it seems that currently the best performance is achievable through high-end supercomputers, such as the IBM Blue Gene/Q [101], where the microprocessor architecture is actually starting to resemble more a GPU than a traditional CPU: the PowerPC A2 chip has 16 in-order cores, each supporting 4 relatively light weight threads and a crossbar on-chip network. A 17th core runs the OS functions and an 18th core is a spare to improve yields or take place of a damaged core. This design gives the PowerPC A2 chip similar performance to power ratio as an NVIDIA Tesla 2090 GPU, making Blue Gene/Q computers very efficient. One of the main advantages of using GPUs (or GPU-like architectures) over traditional serial processors is the increased performance per watt and the possibility to perform simulations on commodity hardware.

5 Wave Function Methods

The stochastic techniques based on Markov chains and the Metropolis algorithm showed great success in the field theory examples above. There are also many-body wave function methods that use the wave function as the central variable and use stochastic techniques for the actual numerical work. These quantum Monte Carlo (QMC) techniques have shown to be very powerful tools for studying electronic structures beyond the mean-field level of for example the density functional theory. A general overview of QMC can be found from [102]. The simplest form of the QMC algorithms is the variational QMC, where a trial wave function with free parameters is constructed and the parameters are optimized, for example, to minimize the total energy [103]. This simple strategy works rather well for various different systems, even for strongly interacting particles in an external magnetic field [104].

There have been some works porting QMC methods to GPUs. In the early work by Amos G. Anderson *et al.* [105], the overall speedup compared to the CPU was rather modest, from three to six, even if the individual kernels were up to 30 times faster. More recently, Kenneth P. Esler *et al.* [106] have ported the QMCPack simulation code to the Nvidia CUDA GPU platform. Their full application speedups are typically around 10 to 15 compared to a quad-core

Xeon CPU. This speedup is very promising and demonstrates the great potential GPU computing has for the QMC methods that are perhaps the computational technologies that are the mainstream in future electronic structure calculations.

There are also many-body wave function methods that are very close to the quantum chemical methods. One example of these is the full configuration interaction method in chemistry that is termed exact diagonalization (ED) in physics. The activities in porting the quantum chemistry approaches to GPU are reviewed in [107], and we try to remain on the physics side of this unclear borderline. We omit, for example, works on the coupled cluster method on the GPU [108]. Furthermore, quantum mechanical transport problems are also not discussed here [109].

Lattice models [110,111] are important for providing a general understanding of many central physical concepts like magnetism. Furthermore, realistic materials can be cast to a lattice model [112]. Few-site models can be calculated exactly using the ED method. The ED method turns out to be very efficient on the GPU [113]. In the simplest form of ED, the first step is to construct the many-body basis and the Hamiltonian matrix in it. Then follows the most time-consuming part, namely the actual diagonalization of the Hamiltonian matrix. In many cases, one is mainly interested in the lowest eigenstate and possibly a few of the lowest excited states. For these, the Lanczos algorithm turns out to be very suitable [113]. The basic idea of the Lanczos scheme is to map the huge but sparse Hamiltonian matrix to a smaller and tridiagonal form in the so-called Krylov space that is defined by the spanning vectors obtained from a starting vector $|x_0\rangle$ by acting with the Hamiltonian as $H^m|x_0\rangle$. Now, as the GPU is very powerful for the matrix-vector product, it is not surprising that high speedups compared to CPUs can be found[113].

6 Outlook

The GPU has made a definite entry into the world of computational physics. Preliminary studies using emerging technologies will always be done, but the true litmus test of a new technology is whether studies emerge where the new technology is actually used to advance science. The increasing frequency of studies that mention GPUs is a clear indicator of this.

From the point of view of high performance computing in computational physics, the biggest challenge facing GPUs at the moment is scaling: in the strong scaling case, as many levels of parallelism as possible inherent in the problem should be exploited in order to reach the best performance with small local subsystems. The basic variables of the model are almost always vectors of some sort, making them an ideal candidate for SIMD type parallelism. This is often achieved with CPUs with a simple compiler flag, which instructs the compiler to look for opportunities to combine independent instructions into vector operations.

Furthermore, large and therefore interesting problems from a HPC point of view are typically composed of a large number of similar variables, be it particles, field values, cells or just entries in an array of numbers, which hints at

another, higher level of parallelism of the problem that traditionally has been exploited using MPI, but is a prime candidate for a data parallel algorithm. Also, algorithmic changes may be necessary to reach the best possible performance: it may very well be that the best algorithm for CPUs is no longer the best one for GPUs. A classic example could be the question whether to use lookup tables of certain variables or recompute them on-the-fly. Typically, on the GPU the flops are cheap making the recomputation an attractive choice whereas the large caches of the CPU may make the lookup table a better option.

On the other hand, MPI communication latencies should be minimized and bandwidth increased to accommodate the faster local solve to help with both weak and strong scaling. As far as we know, there are very few, if any, groups taking advantage of GPUDirect v.2 for NVIDIA GPUs [114], which allows direct GPU-to-GPU communications (the upcoming GPUDirect Support for RDMA will allow direct communications across network nodes) reducing overhead and CPU synchronization needs. Even GPUDirect v.1 helps, as then one can share the *pinned memory* buffers between Infiniband and GPU cards, removing the need to do extra local copies of data. The MPI implementations should also be scaled to fit the needs of the GPUs connected to the node: currently the network bandwidth between nodes seems to be typically about two orders of magnitude lower than the memory bandwidth from the GPU to the GPU memory, which poses a challenge to strong scaling, limiting GPU applicability to situations with relatively large local problem sizes.

Another, perhaps an even greater challenge, facing GPUs and similar systems is the ecosystem: Currently a large portion of the developers and system administrators like to think of GPUs and similar solutions as *accelerators* – an accelerator is a component, which is attached to the main processor and used to speed up certain portions of the code, but as these "accelerators" become more and more agile with wider support for standard algorithms, the term becomes more and more irrelevant as a major part of the entire computation can be done on the "accelerator" and the original "brains" of the machine, the CPU, is mainly left there to take care of administrative functions, such as disk IO, common OS services and control flow of the program.

As single threaded performance has reached a local limit, all types of processors are seeking more performance out of parallelism: more cores are added and vector units are broadened. This trend, fueled by the fact that transistor feature sizes keep on shrinking, hints at some type of convergence in the near future, but exactly what it will look like is anyone's best guess. At least in computational physics, it has been shown already that the scientists are willing to take extra effort in porting their code to take advantage of massively parallel architectures, which should allow them to do the same work with less energy and do more science with the resources allocated to them.

The initial programming effort does raise a concern for productivity: How much time and effort is one willing to spend to gain a certain amount of added performance? Obviously, the answer depends on the problem itself, but perhaps even more on the assumed direction of the industry – a wrong choice may result

in wasted effort if the chosen solution simply does not exist in five years time. Fortunately, what seems to be clear at the moment, is the overall direction of the industry towards higher parallelism, which means that a large portion of the work needed to parallelize a code for a certain parallel architecture will most probably be applicable to another parallel architecture as well, reducing the risk of parallelization beyond the typical MPI level.

The answer to what kind of parallel architectures will prevail the current turmoil in the industry may depend strongly on consumer behavior, since a large part of the development costs of these machines are actually subsidized by the development of the consumer variants of the products. Designing a processor only for the HPC market is too expensive and a successful product will need a sister or at least a cousin in the consumer market. This brings us back to DOOM and other performance-hungry games: it may very well be that the technology developed for the gamers of today, will be the programming platform for the scientists of tomorrow.

Acknowledgements. We would like to thank Kari Rummukainen, Adam Foster, Risto Nieminen, Martti Puska, and Ville Havu for all their support. Topi Siro acknowledges the financial support from the Finnish Doctoral Programme in Computational Sciences FICS. This research has been partly supported by the Academy of Finland through its Centres of Excellence Program (Project No. 251748) and by the Academy of Finland Project No. 1134018.

References

1. Macedonia, M.: The GPU enters computing's mainstream. Computer 36(10), 106–108 (2003)
2. NVIDIA Corporation: NVIDIA CUDA C programming guide, Version 4.2 (2012)
3. McCammon, J.A., Gelin, B.R., Karplus, M.: Dynamics of folded proteins. Nature 267(5612), 585–590 (1977)
4. Tembre, B.L., Cammon, J.M.: Ligand-receptor interactions. Computers & Amp; Chemistry 8(4), 281–283 (1984)
5. Gao, J., Kuczera, K., Tidor, B., Karplus, M.: Hidden thermodynamics of mutant proteins: a molecular dynamics analysis. Science 244(4908), 1069–1072 (1989)
6. Samish, I., MacDermaid, C.M., Perez-Aguilar, J.M., Saven, J.G.: Theoretical and computational protein design. Annual Review of Physical Chemistry 62(1), 129–149 (2011)
7. Berkowitz, M.L., Kindt, J.T.: Molecular Detailed Simulations of Lipid Bilayers, pp. 253–286. John Wiley & Sons, Inc. (2010)
8. Lyubartsev, A.P., Rabinovich, A.L.: Recent development in computer simulations of lipid bilayers. Soft Matter 7, 25–39 (2011)
9. Springel, V., White, S.D.M., Jenkins, A., Frenk, C.S., Yoshida, N., Gao, L., Navarro, J., Thacker, R., Croton, D., Helly, J., Peacock, J.A., Cole, S., Thomas, P., Couchman, H., Evrard, A., Colberg, J., Pearce, F.: Simulations of the formation, evolution and clustering of galaxies and quasars. Nature 435(7042), 629–636 (2005)

10. Chinchilla, F., Gamblin, T., Sommervoll, M., Prins, J.: Parallel N-body simulation using GPUs. Technical report, University of North Carolina (2004)
11. Brodtkorb, A.R., Hagen, T.R., Sætra, M.L.: Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing (2012)
12. Stone, J.E., Hardy, D.J., Ufimtsev, I.S., Schulten, K.: GPU-accelerated molecular modeling coming of age. Journal of Molecular Graphics and Modelling 29(2), 116–125 (2010)
13. Nyland, L., Harris, M., Prins, J.: Fast N-Body Simulation with CUDA. In: GPU Gems 3, ch. 31, vol. 3. Addison-Wesley Professional (2007)
14. Allen, M.P., Tildesley, D.J.: Computer Simulation of Liquids. Clarendon, Oxford (2002)
15. Kipfer, P., Segal, M., Westermann, R.: Uberflow: a GPU-based particle engine. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, HWWS 2004, pp. 115–122. ACM, New York (2004)
16. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for manycore GPUs. Technical report, NVIDIA (2008)
17. Ha, L., Krüger, J., Silva, C.T.: Fast four-way parallel radix sorting on GPUs. Computer Graphics Forum 28(8), 2368–2378 (2009)
18. Anderson, J.A., Lorenz, C.D., Travesset, A.: General purpose molecular dynamics simulations fully implemented on graphics processing units. Journal of Computational Physics 227(10), 5342–5359 (2008)
19. Moreland, K., Angel, E.: The FFT on a GPU. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, HWWS 2003, pp. 112–119. Eurographics Association, Aire-la-Ville (2003)
20. Govindaraju, N.K., Manocha, D.: Cache-efficient numerical algorithms using graphics hardware. Technical report, The University of North Carolina (2007)
21. Gu, L., Li, X., Siegel, J.: An empirically tuned 2d and 3d FFT library on CUDA GPU. In: Proceedings of the 24th ACM International Conference on Supercomputing, ICS 2010, pp. 305–314. ACM, New York (2010)
22. Chen, Y., Cui, X., Mei, H.: Large-scale FFT on GPU clusters. In: Proceedings of the 24th ACM International Conference on Supercomputing, ICS 2010, pp. 315–324. ACM, New York (2010)
23. Ahmed, M., Haridy, O.: A comparative benchmarking of the FFT on Fermi and Evergreen GPUs. In: 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 127–128 (2011)
24. Skeel, R.D., Tezcan, I., Hardy, D.J.: Multiple grid methods for classical molecular dynamics. Journal of Computational Chemistry 23(6), 673–684 (2002)
25. Hardy, D.J., Stone, J.E., Schulten, K.: Multilevel summation of electrostatic potentials using graphics processing units. Parallel Computing 35(3), 164–177 (2009)
26. Goodnight, N., Woolley, C., Lewin, G., Luebke, D., Humphreys, G.: A multigrid solver for boundary value problems using programmable graphics hardware. In: ACM SIGGRAPH 2005 Courses, SIGGRAPH 2005, ACM, New York (2005)
27. McAdams, A., Sifakis, E., Teran, J.: A parallel multigrid poisson solver for fluids simulation on large grids. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2010, pp. 65–74. Eurographics Association, Aire-la-Ville (2010)
28. Meagher, D.: Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. Rensselaer Polytechnic Institute. Image Processing Laboratory (1980)

29. Lefebvre, S., Hornus, S., Neyret, F.: Octree Textures on the GPU. In: GPU Gems 2, ch. 37, vol. 2. Addison-Wesley Professional (2005)
30. Belleman, R.G., Bédorf, J., Zwart, S.F.P.: High performance direct gravitational n-body simulations on graphics processing units ii: An implementation in cuda. New Astronomy 13(2), 103–112 (2008)
31. Hamada, T., Narumi, T., Yokota, R., Yasuoka, K., Nitadori, K., Taiji, M.: 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2009, pp. 62:1–62:12. ACM, New York (2009)
32. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. Journal of Computational Physics 60(2), 187–207 (1985)
33. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. Journal of Computational Physics 73(2), 325–348 (1987)
34. Gumerov, N.A., Duraiswami, R.: Fast multipole methods on graphics processors. Journal of Computational Physics 227(18), 8290–8313 (2008)
35. Darve, E., Cecka, C., Takahashi, T.: The fast multipole method on parallel clusters, multicore processors, and graphics processing units. Comptes Rendus Mécanique 339(2-3), 185–193 (2011)
36. Takahashi, T., Cecka, C., Fong, W., Darve, E.: Optimizing the multipole-to-local operator in the fast multipole method for graphical processing units. International Journal for Numerical Methods in Engineering 89(1), 105–133 (2012)
37. Yokota, R., Barba, L., Narumi, T., Yasuoka, K.: Petascale turbulence simulation using a highly parallel fast multipole method on GPUs. Computer Physics Communications (2012)
38. Götz, A.W., Williamson, M.J., Xu, D., Poole, D., Le Grand, S., Walker, R.C.: Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. generalized Born. Journal of Chemical Theory and Computation 8(5), 1542–1555 (2012)
39. Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. Phys. Rev. 140, A1133–A1138 (1965)
40. Parr, R., Yang, W.: Density-Functional Theory of Atoms and Molecules. International Series of Monographs on Chemistry. Oxford University Press, USA (1994)
41. Payne, M.C., Teter, M.P., Allan, D.C., Arias, T.A., Joannopoulos, J.D.: Iterative minimization techniques for *ab initio* total-energy calculations: molecular dynamics and conjugate gradients. Rev. Mod. Phys. 64, 1045–1097 (1992)
42. Yasuda, K.: Accelerating density functional calculations with graphics processing unit. Journal of Chemical Theory and Computation 4(8), 1230–1236 (2008)
43. Yasuda, K.: Two-electron integral evaluation on the graphics processor unit. Journal of Computational Chemistry 29(3), 334–342 (2008)
44. Ufimtsev, I., Martinez, T.: Graphical processing units for quantum chemistry. Computing in Science Engineering 10(6), 26–34 (2008)
45. Ufimtsev, I.S., Martinez, T.J.: Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation. Journal of Chemical Theory and Computation 4(2), 222–231 (2008)
46. Luehr, N., Ufimtsev, I.S., Martinez, T.J.: Dynamic precision for electron repulsion integral evaluation on graphical processing units (GPUs). Journal of Chemical Theory and Computation 7(4), 949–954 (2011)
47. Ufimtsev, I.S., Martinez, T.J.: Quantum chemistry on graphical processing units. 2. Direct self-consistent-field implementation. Journal of Chemical Theory and Computation 5(4), 1004–1015 (2009)

48. Ufimtsev, I.S., Martinez, T.J.: Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics. Journal of Chemical Theory and Computation 5(10), 2619–2628 (2009)
49. Asadchev, A., Allada, V., Felder, J., Bode, B.M., Gordon, M.S., Windus, T.L.: Uncontracted Rys quadrature implementation of up to G functions on graphical processing units. Journal of Chemical Theory and Computation 6(3), 696–704 (2010)
50. Genovese, L., Ospici, M., Deutsch, T., Méhaut, J.F., Neelov, A., Goedecker, S.: Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. The Journal of Chemical Physics 131(3), 034103 (2009)
51. Genovese, L., Neelov, A., Goedecker, S., Deutsch, T., Ghasemi, S.A., Willand, A., Caliste, D., Zilberberg, O., Rayson, M., Bergman, A., Schneider, R.: Daubechies wavelets as a basis set for density functional pseudopotential calculations. The Journal of Chemical Physics 129(1), 014109 (2008)
52. Kresse, G., Furthmüller, J.: Efficient iterative schemes for *ab initio* total-energy calculations using a plane-wave basis set. Phys. Rev. B 54, 11169–11186 (1996)
53. Maintz, S., Eck, B., Dronskowski, R.: Speeding up plane-wave electronic-structure calculations using graphics-processing units. Computer Physics Communications 182(7), 1421–1427 (2011)
54. Hacene, M., Anciaux-Sedrakian, A., Rozanska, X., Klahr, D., Guignon, T., Fleurat-Lessard, P.: Accelerating VASP electronic structure calculations using graphic processing units. Journal of Computational Chemistry (2012) n/a–n/a
55. Hutchinson, M., Widom, M.: VASP on a GPU: Application to exact-exchange calculations of the stability of elemental boron. Computer Physics Communications 183(7), 1422–1426 (2012)
56. Giannozzi, P., Baroni, S., Bonini, N., Calandra, M., Car, R., Cavazzoni, C., Ceresoli, D., Chiarotti, G.L., Cococcioni, M., Dabo, I., Corso, A.D., de Gironcoli, S., Fabris, S., Fratesi, G., Gebauer, R., Gerstmann, U., Gougoussis, C., Kokalj, A., Lazzeri, M., Martin-Samos, L., Marzari, N., Mauri, F., Mazzarello, R., Paolini, S., Pasquarello, A., Paulatto, L., Sbraccia, C., Scandolo, S., Sclauzero, G., Seitsonen, A.P., Smogunov, A., Umari, P., Wentzcovitch, R.M.: Quantum espresso: a modular and open-source software project for quantum simulations of materials. Journal of Physics: Condensed Matter 21(39), 395502 (2009)
57. Girotto, I., Varini, N., Spiga, F., Cavazzoni, C., Ceresoli, D., Martin-Samos, L., Gorni, T.: Enabling of Quantum-ESPRESSO to petascale scientific challenges. In: PRACE Whitepapers. PRACE (2012)
58. Spiga, F., Girotto, I.: phiGEMM: A CPU-GPU library for porting Quantum ESPRESSO on hybrid systems. In: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 368–375 (February 2012)
59. Wang, L., Wu, Y., Jia, W., Gao, W., Chi, X., Wang, L.W.: Large scale plane wave pseudopotential density functional theory calculations on GPU clusters. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2011, pp. 71:1–71:10. ACM, New York (2011)
60. Jia, W., Cao, Z., Wang, L., Fu, J., Chi, X., Gao, W., Wang, L.W.: The analysis of a plane wave pseudopotential density functional theory code on a GPU machine. Computer Physics Communications 184(1), 9–18 (2013)

61. Enkovaara, J., Rostgaard, C., Mortensen, J.J., Chen, J., Dułak, M., Ferrighi, L., Gavnholt, J., Glinsvad, C., Haikola, V., Hansen, H.A., Kristoffersen, H.H., Kuisma, M., Larsen, A.H., Lehtovaara, L., Ljungberg, M., Lopez-Acevedo, O., Moses, P.G., Ojanen, J., Olsen, T., Petzold, V., Romero, N.A., Stausholm-Møller, J., Strange, M., Tritsaris, G.A., Vanin, M., Walter, M., Hammer, B., Häkkinen, H., Madsen, G.K.H., Nieminen, R.M., Nørskov, J.K., Puska, M., Rantala, T.T., Schiøtz, J., Thygesen, K.S., Jacobsen, K.W.: Electronic structure calculations with GPAW: a real-space implementation of the projector augmented-wave method. Journal of Physics: Condensed Matter 22(25), 253202 (2010)
62. Hakala, S., Havu, V., Enkovaara, J., Nieminen, R.: Parallel Electronic Structure Calculations Using Multiple Graphics Processing Units (GPUs). In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 63–76. Springer, Heidelberg (2013)
63. Castro, A., Appel, H., Oliveira, M., Rozzi, C.A., Andrade, X., Lorenzen, F., Marques, M.A.L., Gross, E.K.U., Rubio, A.: Octopus: a tool for the application of time-dependent density functional theory. Physica Status Solidi (B) 243(11), 2465–2488 (2006)
64. Andrade, X., Alberdi-Rodriguez, J., Strubbe, D.A., Oliveira, M.J.T., Nogueira, F., Castro, A., Muguerza, J., Arruabarrena, A., Louie, S.G., Aspuru-Guzik, A., Rubio, A., Marques, M.A.L.: Time-dependent density-functional theory in massively parallel computer architectures: the Octopus project. Journal of Physics: Condensed Matter 24(23), 233202 (2012)
65. Isborn, C.M., Luehr, N., Ufimtsev, I.S., Martinez, T.J.: Excited-state electronic structure with configuration interaction singles and and Tamm-Dancoff time-dependent density functional theory on graphical processing units. Journal of Chemical Theory and Computation 7(6), 1814–1823 (2011)
66. Peskin, M.E., Schroeder, D.V.: An Introduction to Quantum Field Theory. Westview Press (1995)
67. Crewther, R.J.: Introduction to quantum field theory. ArXiv High Energy Physics - Theory e-prints (1995)
68. Fodor, Z., Hoelbling, C.: Light hadron masses from lattice QCD. Reviews of Modern Physics 84, 449–495 (2012)
69. Göckeler, M., Hägler, P., Horsley, R., Pleiter, D., Rakow, P.E.L., Schäfer, A., Schierholz, G., Zanotti, J.M.: Generalized parton distributions and structure functions from full lattice QCD. Nuclear Physics B Proceedings Supplements 140, 399–404 (2005)
70. Alexandrou, C., Brinet, M., Carbonell, J., Constantinou, M., Guichon, P., et al.: Nucleon form factors and moments of parton distributions in twisted mass lattice QCD. In: Proceedings of The XXIst International Europhysics Conference on High Energy Physics, EPS-HEP 2011, Grenoble, Rhones Alpes France, July 21-27, vol. 308 (2011)
71. McNeile, C., Davies, C.T.H., Follana, E., Hornbostel, K., Lepage, G.P.: High-precision f_{B_s} and heavy quark effective theory from relativistic lattice QCD. Physical Review D 85, 031503 (2012)
72. Rummukainen, K.: QCD-like technicolor on the lattice. In: Llanes-Estrada, F.J., Peláez, J.R. (eds.). American Institute of Physics Conference Series, vol. 1343, pp. 51–56 (2011)
73. Petreczky, P.: Progress in finite temperature lattice QCD. Journal of Physics G: Nuclear and Particle Physics 35(4), 044033 (2008)
74. Petreczky, P.: Recent progress in lattice QCD at finite temperature. ArXiv e-prints (2009)

75. Fodor, Z., Katz, S.D.: The phase diagram of quantum chromodynamics. ArXiv e-prints (August 2009)
76. Montvay, I., Münster, G.: Quantum Fields on a Lattice. Cambridge Monographs on Mathematical Physics. Cambridge University Press, The Edinburgh Building (1994)
77. Rothe, H.J.: Lattice Gauge Theories: An Introduction, 3rd edn. World Scientific Publishing Company, Hackendsack (2005)
78. Gupta, R.: Introduction to lattice QCD. ArXiv High Energy Physics - Lattice e-prints (1998)
79. Egri, G., Fodor, Z., Hoelbling, C., Katz, S., Nogradi, D., Szabo, K.: Lattice QCD as a video game. Computer Physics Communications 177, 631–639 (2007)
80. Schröck, M., Vogt, H.: Gauge fixing using overrelaxation and simulated annealing on GPUs. ArXiv e-prints (2012)
81. Mawhinney, R.D.: The 1 teraflops QCDSP computer. Parallel Computing 25(10-11), 1281–1296 (1999)
82. Chen, D., Christ, N.H., Cristian, C., Dong, Z., Gara, A., Garg, K., Joo, B., Kim, C., Levkova, L., Liao, X., Mawhinney, R.D., Ohta, S., Wettig, T.: QCDOC: A 10-teraflops scale computer for lattice QCD. In: Nuclear Physics B Proceedings Supplements, vol. 94, pp. 825–832 (March 2001)
83. Bhanot, G., Chen, D., Gara, A., Vranas, P.M.: The BlueGene / L supercomputer. Nuclear Physics B - Proceedings Supplements 119, 114–121 (2003)
84. Ammendola, R., Biagioni, A., Frezza, O., Lo Cicero, F., Lonardo, A., Paolucci, P.S., Petronzio, R., Rossetti, D., Salamon, A., Salina, G., Simula, F., Tantalo, N., Tosoratto, L., Vicini, P.: apeNET+: a 3D toroidal network enabling petaFLOPS scale Lattice QCD simulations on commodity clusters. In: Proceedings of The XXVIII International Symposium on Lattice Field Theory, Villasimius, Sardinia Italy, June 14-19 (2010)
85. Shirakawa, T., Hoshino, T., Oyanagi, Y., Iwasaki, Y., Yoshie, T.: QCDPAX-an MIMD array of vector processors for the numerical simulation of quantum chromodynamics. In: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, Supercomputing 1989, pp. 495–504. ACM, New York (1989)
86. Aoki, Y., Fodor, Z., Katz, S.D., Szabó, K.K.: The QCD transition temperature: Results with physical masses in the continuum limit. Physics Letters B 643, 46–54 (2006)
87. Babich, R., Clark, M.A., Joó, B., Shi, G., Brower, R.C., Gottlieb, S.: Scaling lattice QCD beyond 100 GPUs. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2011, pp. 70:1–70:11. ACM, New York (2011)
88. Hasenbusch, M., Jansen, K.: Speeding up the HMC: QCD with clover-improved wilson fermions. Nuclear Physics B Proceedings Supplements 119, 982–984 (2003)
89. Osaki, Y., Ishikawa, K.I.: Domain decomposition method on GPU cluster. In: Proceedings of The XXVIII International Symposium on Lattice Field Theory, Villasimius, Sardinia Italy, June 14-19 (2010)
90. Bonati, C., Cossu, G., D'Elia, M., Incardona, P.: QCD simulations with staggered fermions on GPUs. Computer Physics Communications 183, 853–863 (2012)
91. Winter, F.: Accelerating QDP++ using GPUs. In: Proceedings of the XXIX International Symposium on Lattice Field Theory (Lattice 2011), Squaw Valley, Lake Tahoe, California, July 10-16 (2011)
92. Walk, B., Wittig, H., Dranischnikow, E., Schomer, E.: Implementation of the Neuberger overlap operator in GPUs. In: Proceedings of The XXVIII International Symposium on Lattice Field Theory, Villasimius, Sardinia Italy, June 14-19 (2010)

93. Alexandru, A., Lujan, M., Pelissier, C., Gamari, B., Lee, F.X.: Efficient implementation of the overlap operator on multi-GPUs. ArXiv e-prints (2011)
94. Cardoso, N., Bicudo, P.: SU (2) lattice gauge theory simulations on Fermi GPUs. Journal of Computational Physics 230, 3998–4010 (2011)
95. Cardoso, N., Bicudo, P.: Generating SU(Nc) pure gauge lattice QCD configurations on GPUs with CUDA. ArXiv e-prints (2011)
96. Amado, A., Cardoso, N., Cardoso, M., Bicudo, P.: Study of compact U(1) flux tubes in 3+1 dimensions in lattice gauge theory using GPU's. ArXiv e-prints (2012)
97. Bordag, M., Demchik, V., Gulov, A., Skalozub, V.: The type of the phase transition and coupling values in $\lambda\phi^4$ model. International Journal of Modern Physics A 27, 50116 (2012)
98. Chiu, T.W., Hsieh, T.H., Mao, Y.Y.: Pseudoscalar Meson in two flavors QCD with the optimal domain-wall fermion. Physics Letters B B717, 420 (2012)
99. Munshi, A.: The OpenCL specification, Version 1.2 (2011)
100. Bach, M., Lindenstruth, V., Philipsen, O., Pinke, C.: Lattice QCD based on OpenCL. ArXiv e-prints (2012)
101. IBM Systems and Technology: IBM System Blue Gene/Q – Data Sheet (2011)
102. Foulkes, W.M.C., Mitas, L., Needs, R.J., Rajagopal, G.: Quantum Monte Carlo simulations of solids. Reviews of Modern Physics 73, 33–83 (2001)
103. Harju, A., Barbiellini, B., Siljamaki, S., Nieminen, R., Ortiz, G.: Stochastic gradient approximation: An efficient method to optimize many-body wave functions. Physical Review Letters 79(7), 1173–1177 (1997)
104. Harju, A.: Variational Monte Carlo for interacting electrons in quantum dots. Journal of Low Temperature Physics 140(3-4), 181–210 (2005)
105. Anderson, A.G., Goddard III, W.A., Schröder, P.: Quantum Monte Carlo on graphical processing units. Computer Physics Communications 177(3), 298–306 (2007)
106. Esler, K., Kim, J., Ceperley, D., Shulenburger, L.: Accelerating quantum Monte Carlo simulations of real materials on GPU clusters. Computing in Science and Engineering 14(1), 40–51 (2012)
107. Wölfle, A.W.G., Walker, R.C.: Quantum chemistry on graphics processing units. In: Wheeler, R.A. (ed.). Annual Reports in Computational Chemistry, ch. 2, vol. 6, pp. 21–35. Elsevier (2010)
108. DePrince, A., Hammond, J.: Quantum chemical many-body theory on heterogeneous nodes. In: 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC), pp. 131–140 (2011)
109. Ihnatsenka, S.: Computation of electron quantum transport in graphene nanoribbons using GPU. Computer Physics Communications 183(3), 543–546 (2012)
110. Hubbard, J.: Electron correlations in narrow energy bands. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences 276(1364), 238–257 (1963)
111. Gutzwiller, M.C.: Effect of correlation on the ferromagnetism of transition metals. Physical Review Letters 10, 159–162 (1963)
112. Meredith, J.S., Alvarez, G., Maier, T.A., Schulthess, T.C., Vetter, J.S.: Accuracy and performance of graphics processors: A quantum Monte Carlo application case study. Parallel Computing 35(3), 151–163 (2009)
113. Siro, T., Harju, A.: Exact diagonalization of the Hubbard model on graphics processing units. Computer Physics Communications 183(9), 1884–1889 (2012)
114. NVIDIA Corporation: NVIDIA GPUDirect^{TM} Technology (2012)