CHAPTER 2

Quantum Chemistry on Graphics Processing Units

Andreas W. Götz[1], **Thorsten Wölfle**[1,2], and **Ross C. Walker**[1]

Contents

Abstract

We report on the current status of algorithm development and software implementations for acceleration of quantum chemistry and computational condensed matter physics simulations on graphics processing units (GPUs) as documented in the peer-reviewed literature. We give a general overview of programming techniques and concepts that should be considered when porting scientific software to GPUs. This is followed by a discussion of Hartree-Fock and density functional theory, wave function-based electron correlation methods and quantum Monte Carlo in which we outline the underlying problems and present the approaches which aim at exploiting the performance of the massively parallel GPU hardware. We conclude with a

[1] San Diego Supercomputer Center, University of California San Diego, La Jolla, CA, USA

[2] Lehrstuhl für Theoretische Chemie, Universität Erlangen, Erlangen, Germany

Annual Reports in Computational Chemistry, Volume 6
ISSN: 1574-1400, DOI 10.1016/S1574-1400(10)06002-0

critical assessment of the present state of the field and discuss future directions that are likely to be taken.

Keywords: quantum chemistry; density functional theory; Hartree–Fock theory; Møller–Plesset perturbation theory; quantum Monte Carlo; graphics processing units; CUDA; NVIDIA; ATI; accelerator

1. INTRODUCTION

Commodity graphics processing units (GPUs) are becoming increasingly popular to accelerate molecular and condensed matter simulations due to their low cost and potential for high performance when compared with central processing units (CPUs). In many instances, classical approximations are very successful for such simulations. However, a large number of problems of contemporary nano-, bio-, or materials science require a quantum mechanical description of the electronic structure [1–3]. This chapter provides an overview of recent developments within quantum chemistry and computational condensed matter physics that utilize accelerator hardware for this purpose.

Quantum chemistry and solid-state physics codes implement relatively complex algorithms [4]. The challenge in using GPUs lies in adapting these algorithms to take advantage of their specialized hardware. A successful GPU implementation requires, for example, a careful consideration of the memory hierarchy in order not to expose memory access latency [5]. When using single-precision GPUs, the numerical accuracy is a central issue because six to seven significant figures are frequently insufficient to match the accuracy of the underlying theoretical model, that is, to achieve "chemical accuracy" of 1 kcal mol^{-1}.

Finally, care should be taken to allow for a coevolution of the code with the hardware. There are two general strategies for an implementation. First, a complete reimplementation of existing functionality into a new software package. The most common way, however, is to incrementally include GPU kernels for the computationally intensive parts of existing software packages. The latter approach has the advantage of retaining the full functionality of software packages that in many cases have evolved over several decades.

This chapter begins with a brief introduction to the general concepts that have to be considered in order to successfully port scientific software to GPUs. The rest of this chapter is structured according to the different theoretical models commonly used in quantum chemistry, beginning with density functional theory (DFT) in Section 3 which also covers Hartree–Fock (HF) theory. Section 4 deals with *ab initio* electron correlation methods while Section 5 discusses quantum Monte Carlo (QMC). Each of these sections contains an overview of the critical parts of the underlying theory followed by a presentation and analysis of approaches that have been taken to accelerate the computationally intensive parts on GPUs. Section 6 summarizes the present state of GPU implementations for quantum chemistry and finishes with general conclusions on trends to be expected in the foreseeable future.

2. SOFTWARE DEVELOPMENT FOR GRAPHICS PROCESSING UNITS

An excellent introduction to software development for GPUs including a discussion of the hardware and its historic development can be found in the book of Kirk and Hwu [5]. In order to be able to write software which runs efficiently on GPUs, it is necessary to have an understanding of the characteristics of the GPU hardware architecture.

A GPU is an example of a massively parallel stream-processing architecture which uses the single-instruction multiple data (SIMD) vector processing model. Typical GPUs contain many arithmetic units which are arranged in groups that share fast access memory and an instruction unit. The high density of arithmetic units, however, comes at the expense of larger cache sizes and control units. The NVIDIA GeForce 8800 GTX GPU which was released in late 2006, for example, consists of 16 sets of streaming multiprocessors (SMs), each of which is composed of eight scalar processors (ScaPs). Each SM operates independently of the other SMs and at any given clock cycle, each ScaP within an SM executes the same instruction but for different data. Due to this intrinsic parallelization, a GPU can outperform a standard CPU for tasks which exhibit a dense level of data parallelism. Successful approaches in GPU programming therefore require exposing the data parallelism in the underlying problem.

Each SM has access to four different types of on-chip memory with high bandwidth. In the case of the NVIDIA GeForce 8800 GTX, these are 1024 local registers per ScaP, shared memory (cache) of 16 kilobytes (KB), read-only constant cache of 8 KB to speed up reads from the constant memory space, and read-only texture cache of 8 KB to speed up reads from the texture memory space. In addition, a large, noncached off-chip graphics card memory is available. This memory, however, has a high latency of approximately 500 GPU cycles. Applications on a GPU are organized into streams and kernels. The former represent blocks of data while the latter execute operations on the data. Before a GPU kernel is executed, the CPU must copy required data to the GPU memory. To maximize the speedup of the implemented kernels, the algorithm has to be adapted to the underlying hardware architecture-dependent features like memory layout. Copy operations between main memory and graphics card memory, for example, should be avoided because access to the main memory has a high latency on the order of hundreds of GPU cycles. One of the main problems when programming GPUs is the limited size of working memory (registers, caches) which are available on chip. A large number of parallel threads should therefore be run concurrently to hide the latency of the registers and the shared and global memory and avoid pipeline stalls.

It is important to realize that many of these considerations are not only important for GPU programming. The arrangement of data in a data-parallel fashion, for example, is also important for parallel programming of distributed memory architectures, which are found in most of today's standard CPU clusters. Thus many of the techniques employed to improve the parallel efficiency of quantum chemistry codes are also applicable to GPUs. The same holds for the optimization of memory access patterns. A general example for a portable algorithm is the fastest fourier transform in the west (FFTW) Fourier transform library which reaches optimal performance on the target platform by using a divide-and-conquer strategy [6].

Early use of GPUs required one to describe a problem to be solved in terms of a graphics pipeline employing either OpenGL or DirectX graphics programming languages. This complexity made general purpose computation on GPUs a research topic. However, with the release of NVIDIA's compute unified device architecture (CUDA) [7] and ATI's Stream [8] application programming interfaces (APIs), implementations of algorithms for GPUs using a relatively simple extension of the standard C language have become possible. A detailed overview of the hardware and CUDA and Stream APIs can be found on the NVIDIA [9] and ATI [8] homepages, respectively. In addition, high abstraction subroutine libraries are available that provide algorithms for commonly used problems in quantum chemistry and solid-state physics such as Fourier transforms (CUFFT) [10] and linear algebra (CUBLAS, MAGMA) [11,12].

The first generation of GPUs to support CUDA, such as the NVIDIA Geforce 8800 GTX, only featured 32-bit single-precision (SP) arithmetics and thus was of only limited use for quantum chemistry. Major efforts had to be made to deal with roundoff errors resulting from the lack of 64-bit double-precision (DP) data types. The second generation of GPUs introduced the missing 64-bit arithmetics, albeit only at an eighth of the SP performance. GPU cards dedicated to general purpose computing such as the NVIDIA Tesla C1060, which also supports large amounts of up to 4 gigabytes (GB) onboard memory, were introduced. The low speed of the DP arithmetics and missing features such as error-correcting code (ECC), however, still hamper widespread acceptance of this generation of GPUs for scientific computing as compared to multi-socket CPUs. The third generation of GPUs (such as the NVIDIA Fermi) will solve some of the major problems of the earlier models. Most importantly, DP support will be included at only half the speed of SP arithmetics. The availability of a global address space and 64-bit support will help to address the memory requirement to solve larger problems and support multiple GPUs in an easier and more transparent fashion. Access to CPU main memory will remain slow, however, because the data transfer takes place over the peripheral component interconnect (PCI) bus.

3. KOHN–SHAM DENSITY FUNCTIONAL AND HARTREE–FOCK THEORY

Due to its excellent balance between accuracy and computational cost, Kohn–Sham density functional theory (KS-DFT) [13,14] is usually the method of choice to investigate electronic ground states and their properties in chemistry and solid-state physics [15,16]. Hartree–Fock (HF) wavefunctions, on the other hand, are the starting point for *ab initio* electron correlation methods [4,15] which are discussed in Section 4.

There are two major computational bottlenecks in KS-DFT and HF calculations [15]: evaluation of the KS (or Fock) matrix elements and solution of the self-consistent field (SCF) equations. The latter requires diagonalization of the Fock

Table 1 Summary of the capabilities and performance of GPU-based KS-DFT and HF implementations published to date

Authors	$l_{\max}$	ERIs	$J_{\mu\nu}$	$K_{\mu\nu}$	$V_{\mu\nu}^{\mathrm{XC}}$	∇E	Parallel[a]	Speedup[b]
Yasuda [18,27]	p	Yes	Yes	No	Yes	No	No	10
Ufimtsev and Martinez [21,23,25]	p	Yes	Yes	Yes	No	Yes	Yes	100
Asadchev et al. [26]	g	Yes	No	No	No	No	No	25
Brown et al. [30,31]	f	No	Yes[c]	No	Yes	Yes	Yes	15[d]

[a] Support for parallelization across multiple GPUs.
[b] Estimates for one GPU as compared to one CPU.
[c] Contribution due to Poisson density fitting via numerical quadrature.
[d] Using 12 ClearSpeed xe620 accelerator cards.

matrix which eventually dominates the computational cost for very large calculations. This topic has not been extensively discussed in the GPU literature but could potentially be tackled with alternatives to diagonalization as employed in linear scaling approaches to electronic structure methods [17].

The computational effort for the formation of the KS (or Fock) matrix is dominated by the evaluation of the two-electron repulsion integrals (ERIs) which are required for the Coulomb and exact-exchange contributions and, in the case of DFT, also the numerical quadrature of the exchange-correlation (XC) contribution. Efforts to accelerate these steps are summarized in Table 1 and reviewed in the remainder of this section.

3.1 Electron repulsion integrals

The ERIs which are required in quantum chemistry are given as

$$(\mu\nu|\kappa\lambda) = \int \mathrm{d}\mathbf{r}\,\mathrm{d}\mathbf{r}' \frac{\phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r})\phi_\kappa(\mathbf{r}')\phi_\lambda(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}, \tag{1}$$

where ϕ_μ are basis functions that are usually chosen to be atom-centered Gaussian functions. In general, these basis functions are contracted, that is, linear combinations of primitive Gaussian functions χ_p and the ERIs become

$$(\mu\nu|\kappa\lambda) = \sum_{pqrs} d_{\mu p} d_{\nu q} d_{\kappa r} d_{\lambda s} (pq|rs). \tag{2}$$

Formally, $\mathcal{O}(N^4)$ of these ERIs need to be evaluated, where N denotes the size of the molecule under consideration. Although for large systems most of the integrals are zero or negligible, the asymptotic scaling remains $\mathcal{O}(N^2)$ and the sheer number of ERIs that need to be calculated represents a major computational bottleneck. Many different algorithms have been devised for the calculation of these ERIs and their efficiency depends on the contraction length and angular momentum quantum number of the basis functions involved [4]. CPU-based quantum chemistry codes therefore implement several ERI algorithms and make use of the best method for a given type of ERI.

From the ERIs, the Coulomb and exact-exchange contributions to the KS (or Fock) matrix are obtained as

$$J_{\mu\nu} = \sum_{\kappa\lambda} P_{\kappa\lambda}(\mu\nu|\kappa\lambda), \qquad K_{\mu\nu} = \sum_{\kappa\lambda} P_{\kappa\lambda}(\mu\kappa|\nu\lambda), \tag{3}$$

where $P_{\mu\nu}$ are elements of the density matrix. As is common in direct SCF methods, by combining Eqs. (1) and (3), the contributions to the KS (or Fock) matrix can be evaluated directly such that the contracted ERIs never need to be explicitly calculated and stored in memory.

Yasuda was the first to realize the potential of GPUs for the acceleration of ERI calculations [18]. In his work, the major problems hindering algorithm development on GPUs are addressed and the results for the calculation of the Coulomb contribution to the KS matrix with s- and p-type basis functions are presented for a CUDA implementation. Although it is not the most efficient algorithm for ERIs over basis functions with low angular momentum quantum number, the Rys quadrature [19] scheme was chosen. Due to its low memory requirements, this scheme allows one to maximize the load balance of the GPU's SMs. A new interpolation formula for the roots and weights of the quadrature was proposed which is particularly suitable for SIMD processing, and an error analysis for the quadrature was given. A mixed-precision (MP) CPU/GPU scheme was introduced which calculates the largest ERIs (prescreened via the Schwarz integral bound and an adjustable threshold) in DP on the CPU and the remaining ERIs in SP on the GPU such that the absolute error in the calculated ERIs can be controlled. This, together with data accumulation (Coulomb matrix formation) via 48-bit multiprecision addition (which can be implemented in software for GPUs without DP support), leads to accurate DFT SCF energies while the errors are of the order of 10^{-3} au (around 1 kcal mol^{-1}) if all ERIs are calculated on the GPU. The contributions to the Coulomb matrix are directly computed from the uncontracted ERIs in a SIMD fashion on the GPU which avoids the problem of having to transfer the large amount of ERIs from GPU to CPU memory. Instead, only the density and Coulomb matrix have to be transferred. If all ERIs are evaluated on the GPU (NVIDIA GeForce 8800 GTX), speedups around one order of magnitude have been observed for the formation of the Coulomb matrix for molecules as big as valinomycin (168 atoms) with a 6-31G basis set as compared to a conventional implementation running on an Intel Pentium 4 CPU with 2.8 GHz [18]. If part of the ERIs are calculated on the CPU to reduce the error in the total energy to 10^{-6} au (less than 10^{-3} kcal mol^{-1}), the speedup drops to around three. However, as Yasuda states, there is room for improvement in the performance, for example, through pipelining and also potentially by exploiting the DP functionality of current and future GPUs.

Ufimtsev and Martinez (UM) have also developed CUDA kernels for the calculation of ERIs and Fock matrix formation involving s- and p-type basis functions on GPUs [20,21]. They opted for the McMurchie–Davidson [22] scheme because it requires relatively few intermediates per integral resulting in a low memory requirement, similar to the Rys quadrature. Three different mappings of the computational work to thread blocks have been tested which result in different load balancing and data reduction overhead and the ERI kernels have carefully been optimized accordingly [21]. If the Fock matrix contributions are directly evaluated from the primitive ERIs, it becomes most efficient to assign the calculation of each primitive ERI batch (i.e., all ERIs over basis functions with magnetic quantum numbers for the given angular momentum quantum numbers) to one thread, independent of the contraction length of the basis functions. In order to maximize load balancing, the integral batches are presorted into blocks of angular momentum classes [21] and within these blocks according to their magnitude [23]. As in Yasuda's work [18], the Fock matrix elements are directly computed on the GPU but pre- and postprocessing are done on the CPU. This approach has been parallelized over multiple GPUs [23].

HF SCF calculations with a 3-21G and 6-31G basis set using UM's implementation and an NVIDIA GTX280 card can be more than 100 times faster than the quantum chemistry program package GAMESS [24] on a single 3.0 GHz Intel Pentium D CPU [25]. For small- and medium-sized molecules, most of the time is spent in the Fock matrix formation on the GPU. However, for large molecules such as olestra (453 atoms, 2131 basis functions), the linear algebra (LA) required for the solution of the SCF equations starts to become a bottleneck, requiring as much as 50% of the Fock matrix computation time (LA performed on the GPU using CUBLAS). A parallel efficiency of over 60% was achieved on three NVIDIA GeForce 8800 GTX cards as compared to the use of only one graphics accelerator. Two points should be mentioned here. First, the limitation to s- and p-type functions results in small integral blocks that can be treated entirely in shared memory and registers which means that the ratio of computation to memory access is high. This situation will change for basis functions with higher angular momentum quantum numbers. Furthermore, the Rys quadrature [19] which was used by GAMESS in these comparisons is a legacy Fortran implementation that underperforms on modern CPUs [26]. ERI algorithms which are more efficient on CPUs do exist and less favorable GPU speedups should be observed for comparisons against implementations of these algorithms which are optimized for performance on modern CPUs.

The error in the SCF energies obtained with UM's code due to the use of SP arithmetics quickly exceeds 10^{-3} au (chemical accuracy, less than 1 kcal mol^{-1}) for larger molecules [23]. However, ERI evaluation in SP and data accumulation in DP, which can be performed on newer GPUs with negligible additional computational cost, improve the accuracy to this level in all investigated cases. In addition, error compensation in relative energies was observed, presumably due to cancellation of contributions of large ERIs. For larger molecules, however, computation of the larger ERIs in DP will be required, as has been extensively discussed before by Yasuda [18].

UM have also implemented the calculation of the Coulomb and exact-exchange contributions to the analytical HF energy gradients with s- and p-type basis functions on GPUs [25]. Using the 3-21G basis set, a speedup between 6 for small molecules and over 100 for larger molecules (olestra) has been obtained running in parallel on a system equipped with two NVIDIA GTX295 cards (each of which has two GPUs) and an Intel Core2 quad-core 2.66 GHz CPU. Reference was again made to GAMESS, running in parallel on all four CPU cores. Using the mixed SP/DP approach discussed above, the root mean squared error in the forces is distributed around 10^{-5} au, which is close to typical convergence thresholds for geometry optimizations. Geometry optimization of a helical hepta-alanine was shown to lead to an optimized structure in good agreement with GAMESS results with an error in the final energy as low as 0.5 kcal mol^{-1}. Good energy conservation was shown for an HF Born–Oppenheimer molecular dynamics simulation of an $H_3O+(H_2O)_{30}$ cluster with the 6-31G basis set in the microcanonical ensemble using the velocity Verlet algorithm with a time step of 0.5 fs. An energy drift of 0.022 kcal mol^{-1} ps^{-1} was observed over a simulation time of 20 ps.

Recently, Asadchev et al. presented algorithms and a CUDA implementation for the calculation of uncontracted ERIs including up to g-type functions [26]. The Rys quadrature [19] was chosen which, in addition to its low memory footprint, is efficient for integrals with higher order angular momentum. The major problem is that, unlike numerical LA kernels, the quadrature has very complex memory access patterns which span a large data set and depend on the particular ERI class being evaluated. As an example, an (ff|ff) ERI shell block requires 5376 floating-point numbers for intermediate quantities which are reused multiple times and 10^4 floating-point numbers for the final ERIs [26]. With DP this corresponds to 123,008 bytes, which is much larger than cache sizes available on GPUs. Therefore, these intermediates must be stored and loaded from the device memory as required and it becomes mandatory to arrange the parallel calculation of the ERIs in such a way that these memory loads are minimized. For this purpose, integrals in a shell block are reordered such that intermediates can be reused as often as possible. Another problem is the large amount of code required to cover all possible cases of integral types in an efficient manner. The authors therefore adopted a template-based approach in which all cases can be generated from a single template in an automated fashion.

The performance of these GPU ERI kernels was tested on NVIDIA GeForce GTX 275 and NVIDIA Tesla T10 cards and compared to the performance of the ERI evaluation with the Rys quadrature as implemented in GAMESS (which, as noted above, underperforms on modern CPUs) [26]. While the CPU code achieves around 1 GFLOPS (giga floating point operations per second), the GPUs achieve around 25 GFLOPS in DP and 50 GFLOPS in SP, which is approximately 30% of the theoretically possible DP peak performance. The difference between performance in SP and DP is approximately a factor of 2 which shows that the computations are memory bound rather than compute bound. No timings are given for the data transfer between GPU memory and main memory apart from stating that it takes several times longer than the actual execution time of the ERI kernels. It is clear that, in order to retain the speed advantage of the ERI evaluation on the GPU, processing of the ERIs (e.g., Fock matrix formation) must be implemented on the GPU device, as well.

3.2 Numerical exchange-correlation quadrature

In the generalized gradient approximation (GGA) to DFT, the XC potential depends on the electron density ρ and its gradient $\nabla\rho$ and is a complicated function in three-dimensional space. This makes an analytical solution of the XC integrals impossible and numerical quadrature is used to compute the XC matrix elements,

$$V_{\mu\nu}^{\mathrm{XC(GGA)}} = \int d\mathbf{r}\, \phi_\mu(\mathbf{r})\nu_{\mathrm{XC}}^{\mathrm{GGA}}(\mathbf{r})\phi_\nu(\mathbf{r}) \approx \sum_k w_k \phi_\mu(\mathbf{r}_k)\nu_{\mathrm{XC}}^{\mathrm{GGA}}(\mathbf{r}_k)\phi_\nu(\mathbf{r}_k), \tag{4}$$

where $\mathbf{r}_k$ are the quadrature points and w_k the corresponding weights.

The numerical XC quadrature is perfectly suited for parallelization and Yasuda was the first to exploit GPUs for this purpose [27]. He adopted a strategy in which the computationally less demanding steps in the quadrature (grid generation, evaluation of $\nu_{\mathrm{XC}}^{\mathrm{GGA}}$ on the grid points) are done in DP on the CPU while the expensive steps are done on the GPU. These are the evaluation of ρ and $\nabla\rho$ on the grid points and the summation of Eq. (4) which can be formulated as matrix-vector multiplications and dot products. Both steps are organized in batches of grid points and nonnegligible basis functions that are small enough to be kept entirely in shared memory. Although in this way some of the basis function values on the grid points must be recalculated, this is more than compensated for by the low latency of the shared memory.

In order to deal with roundoff errors due to the use of SP floating-point numbers on the GPU, Yasuda introduced a scheme in which the XC potential is approximated with a model potential $\nu_{\mathrm{XC}}^{\mathrm{model}}$ which is chosen such that its matrix elements can be calculated analytically. This is done in DP on the CPU while the GPU is used for calculating the correction, that is, for the numerical quadrature of the matrix elements of $\Delta v_{\mathrm{XC}} = \nu_{\mathrm{XC}}^{\mathrm{GGA}} - \nu_{\mathrm{XC}}^{\mathrm{model}}$. Without the model potential, errors in the total energy of valinomycin with a 3-21G or 6-31G basis set and the PW91 [28] XC functional are close to 10^{-4} au. With the model potential approach, the error is reduced to 10^{-5} au which is sufficient for most purposes. A speedup of approximately 40 is observed with an NVIDIA GeForce 8800 GTX graphics card as compared to a conventional implementation running on an Intel Pentium 4 CPU with 2.8 GHz. This translates into a speedup of around five to ten as compared to more modern CPUs.

3.3 Density-fitted Poisson method

Brown et al. have presented a different heterogeneous approach to accelerate DFT, combining ClearSpeed accelerator cards [29] in parallel with a host CPU [30,31]. The ClearSpeed accelerator hardware is a compute-oriented stream architecture with raw performance comparable to that of modern GPUs while offering support for DP. Just as for GPUs, an efficient use of this hardware requires fine-grained parallelization with a large number of lightweight threads and any algorithm developed for these accelerators will map well onto GPUs. By using the Poisson density fitting method, all bottlenecks of a DFT calculation could be shifted into finely parallelizable numerical quadrature. Density fitting [32,33], also called resolution-of-identity (RI) [34] Coulomb method, is used to avoid the need to calculate the four-index ERIs of Eq. (1). Instead, the Coulomb contributions to the KS matrix are obtained from three-center ERIs $(\mu\nu|\alpha)$, where φ_α are auxiliary density fitting basis functions. As a result, the formal scaling of this step becomes $\mathcal{O}(N^3)$ and the prefactor is reduced. The auxiliary basis set can be chosen to consist of a few atom-centered Gaussian functions augmented with Poisson functions (obtained by applying the Poisson operator $\hat{p} = -(4\pi)^{-1}\nabla^2$ to atom-centered Gaussian functions) whereby the majority of the three-index ERIs is replaced with short-ranged three-index overlap integrals $(\mu\nu,\alpha) = \int \mathrm{d}\mathbf{r}\, \phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r})\varphi_\alpha(\mathbf{r})$. This leads to a further reduction of the prefactor. Furthermore, these overlap integrals can be calculated by numerical quadrature. However, to maintain numerical stability in the SCF procedure, a higher accuracy than provided by default XC quadrature grids is required, thus increasing the number of grid points.

The implementation, which is not restricted to basis functions with low angular momentum quantum numbers, passes only information about the numerical quadrature grid, the basis functions, the KS matrix, and the density matrix between the accelerator cards and the host system. The numerical quadrature of the XC contribution and the Coulomb contribution due to the integrals $(\mu\nu,\alpha)$ is done on the accelerator cards in batches of grid points such that all computations can be done within the cache memory of the accelerator cards. All other parts of the DFT calculation are performed on the host CPU. Compared to an implementation with analytical evaluation of the integrals $(\mu\nu,\alpha)$ running on one core of a dual core AMD Opteron 2218 CPU with 2.6 GHz, a speedup between 7 and 15 was observed with 12 ClearSpeed xe620 cards for SCF single-point [30] and gradient [31] calculations. The calculations were run for molecules of the size between chorismate (24 atoms) and an alanine helix consisting of 12 monomers (123 atoms) with 6-31G* and cc-pVTZ and corresponding density fitting basis sets. There is further room for improvement, for example, by implementing prescreening which is missing so far. However, work done on the host is already becoming a bottleneck and needs to be addressed. The diagonalization, for example, takes approximately 30% of the total runtime.

3.4 Density functional theory with Daubechies wavelets

Another effort in the physics community should be mentioned here. The BigDFT software [35] is based on Daubechies wavelets instead of Gaussian basis functions and offers support within the CUDA programming framework. It was shown to achieve a high parallel efficiency of 90% on parallel computers in which the cross-sectional bandwidth scales well with the number of processors. It uses a parallelized hybrid CPU/GPU programming model and compared to the full CPU implementation, a constant speedup of up to six was achieved with the GPU-enabled version [35].

4. *AB INITIO* ELECTRON CORRELATION METHODS

The quantum chemist's traditional way to approximate solutions of the electronic Schrödinger equation is so-called *ab initio*, wave function-based electron correlation methods. These methods improve upon the HF mean-field approximation by adding many-body corrections in a systematic way [15]. As of the time of this writing, efforts to accelerate *ab initio* calculations with GPUs are scarce. However, it is expected that this will change in the near future because these methods are of critical importance whenever higher accuracy is required than what can be achieved by DFT or for types of interactions and properties for which DFT breaks down.

4.1 Resolution-of-identity second-order Møller–Plesset perturbation theory

Second-order Møller–Plesset perturbation theory (MP2) is the computationally least expensive and most popular *ab initio* electron correlation method [4,15]. Except for transition metal compounds, MP2 equilibrium geometries are of comparable accuracy to DFT. However, MP2 captures long-range correlation effects (like dispersion) which are lacking in present-day density functionals. The computational cost of MP2 calculations is dominated by the integral transformation from the atomic orbital (AO) to the molecular orbital (MO) basis which scales as $\mathcal{O}(N^5)$ with the system size. This four-index transformation can be avoided by introduction of the RI integral approximation which requires just the transformation of three-index quantities and reduces the prefactor without significant loss in accuracy [36,37]. This makes RI-MP2 the most efficient alternative for small- to medium-sized molecular systems for which DFT fails.

Aspuru-Guzik and coworkers have worked on accelerating RI-MP2 calculations [38,39]. They exploited the fact that the step which dominates the computational cost of an RI-MP2 calculation essentially consists of matrix multiplications to generate the approximate MO integrals from the half-transformed three-index integrals $B_{ia,P}$,

$$(ia|jb) \approx \sum_P B_{ia,P} B_{jb,P}. \tag{5}$$

Here, i, j (a, b) label occupied (virtual) MOs and P labels auxiliary basis functions. CPU implementations proceed by multiplying a matrix of size $N_{virt} \times N_{aux}$ (number of virtual orbitals $\times$ number of auxiliary basis functions) against its transpose for each pair ij of occupied orbitals.

To take full benefit of GPUs for these matrix multiplications, the matrices have to be larger than a given threshold to minimize the impact of the bus latency when transferring the matrices from the CPU to the GPU memory. Depending on the system size (number of atoms, size of basis sets employed), this is achieved by treating several pairs ij of occupied orbitals together [38].

For the multiplication of general matrices whose size is too large to be held in the onboard memory of the GPU, a library has been developed [39,40]. As established for standard parallel matrix multiplications, this library uses a two-dimensional decomposition. Partial matrix multiplications of these blocks are performed on the GPU with CUBLAS routines and the results are accumulated on the CPU. To improve the numerical accuracy, a heterogeneous computing model is employed in which numerically large contributions to the final result are computed and accumulated on a DP device (in general the CPU) and the remaining small contributions are efficiently treated by the SP GPU device. It was shown that errors can be reduced by an order of magnitude in exchange for a moderate performance decrease with this MP approach.

Compared to the standard CPU implementation, speedups of 13.8, 10.1, and 7.8 were obtained on an NVIDIA Tesla C1060 GPU equipped with 4 GB of memory for the 168-atom molecule valinomycin in SP, MP, and DP, respectively. The corresponding correlation energy error is $-10.0\,\text{kcal mol}^{-1}$, $-1.2\,\text{kcal mol}^{-1}$, and essentially zero, respectively [39]. While the largest speedup can be obtained by performing the matrix multiplications entirely in SP, the resulting error is larger than acceptable for chemical accuracy. It is therefore inevitable to put up with some performance penalty for the sake of accuracy. It was shown that the ERI evaluation becomes computationally as expensive as the integral transformation [38]. We therefore anticipate a combination with the approaches discussed in Section 3 for the ERI evaluation.

5. QUANTUM MONTE CARLO

Quantum Monte Carlo (QMC) [41] is one of the most accurate methods for solving the time-independent Schrödinger equation. As opposed to variational *ab initio* approaches, QMC is based on a stochastic evaluation of the underlying integrals. The method is easily parallelizable and scales as $\mathcal{O}(N^3)$, however, with a very large prefactor.

Anderson et al. have shown [42] how to accelerate QMC calculations by executing CUDA kernels that are explicitly optimized for cache usage and instruction-level parallelism for the computationally intensive parts on a GPU. These are the basis function evaluation on grid points and, similar to the numerical XC quadrature and RI-MP2, matrix multiplications. The Kahan Summation Formula to improve the accuracy of GPU matrix multiplications was explored which was necessary because of the lack of fully compliant IEEE floating-point implementations on GPUs in 2007. For small molecules with 8–28 atoms (32–152 electrons and 80–516 basis functions), approximately fivefold speedup was obtained using an NVIDIA GeForce 7800 GTX graphics card as compared to an optimized implementation running on an Intel Pentium 4 CPU with 3 GHz.

Meredith et al. have used an implementation of the quantum cluster approximation on SP GPUs to study the effect of disorder on the critical temperature for superconductivity in cuprates with a two-dimensional Hubbard model on a regular lattice [43]. Trivial modifications to the code base were made, performing matrix multiplications on the GPU using the CUBLAS library. Attempts to increase the performance by circumventing the data transfer bottleneck and implementing the remaining data manipulations on the GPU instead of the CPU resulted in a performance loss for all but the largest problem size that was investigated. The simple reason is that smart algorithms that can be implemented efficiently on CPUs do not map well onto GPU architectures or, in other words, the GPU has to do more work to achieve the same result. For the largest problem size studied, a fivefold speedup was observed running on a cluster with 32 AMD Opteron 2.6 GHz CPUs and 32 NVIDIA 8800 GTX graphics cards as compared to using only the CPUs in parallel. Sufficient accuracy for scientifically meaningful results within the employed model was proven by comparison to DP results obtained on a CPU.

6. CONCLUDING REMARKS

Quantum chemistry software that exploits the capabilities of modern GPUs has only recently started to emerge. Significant parts of these initial efforts have been devoted to minimize errors caused by the lack of DP support on older GPUs. The advent of next-generation GPUs that support DP arithmetics at a peak performance of only a factor of 2 less than that of SP will make these special approaches obsolete. At the same time, future developments will be greatly facilitated.

From the literature, one can observe that in order to achieve good results in programming with GPUs it is often necessary to write GPU-only versions of the code. One typically has to abandon many of the smart optimizations that have been developed over the years for CPUs and expensive copy operations from the CPU to the GPU memory have to be minimized.

With careful work, it is possible to achieve speedups which should allow researchers to perform calculations that otherwise would require large and expensive CPU clusters. However, the nature of GPU programming is such that significant effort is still required to make effective use of GPUs. These complexities are the reason that the quantum chemistry software that is available for GPUs at the time of this writing is still in its infancy and not yet ready for general use. GPU implementations that are capable of full HF and DFT calculations, for example, are still restricted to s- and p-type basis functions. HF calculations are not of much practical use by themselves but only as starting point for correlated *ab initio* methods which require basis functions with high angular momentum quantum numbers. Similarly, meaningful DFT calculations have to use polarization functions which means that even for simple organic molecules or biomolecules without metal atoms at least d-type functions are required. While GPU-based ERI implementations for high angular momentum basis functions have been developed, these still have to be incorporated into software capable of performing *ab initio* or DFT calculations.

Up to now only energies and gradients have been considered which allows for explorations of potential energy surfaces. However, a variety of other quantum chemistry applications would also benefit from the computational power that GPUs provide. Of high interest for the researcher are static and dynamic molecular response properties. Frequently, these require a higher computational effort than energy and gradient evaluations. We therefore expect to see developments in this area soon.

We are looking forward to exciting new developments of quantum chemistry software for GPUs accompanied by ground-breaking applications in the near future.

ACKNOWLEDGMENTS

This work was supported in part by grant 09-LR-06-117792-WALR from the University of California Lab Fees program and grant XFT-8-88509-01/DE-AC36-99GO10337 from the Department of Energy to RCW.

REFERENCES

1. Clary, D.C. Quantum chemistry of complex systems. Science 2006, 314(5797), 265–6.
2. Carter, E.A. Challenges in modeling materials properties without experimental input. Science 2008, 321(5890), 800–3.
3. Reiher, M. (ed.) Atomistic Approaches in Modern Biology, Topics in Current Chemistry, Springer, Heidelberg, 2007.
4. Helgaker, T., Jørgensen, P., Olsen, J. Molecular Electronic-Structure Theory, Wiley, West Sussex, England, 2000.
5. Kirk, D.B., Hwu, W.W. Programming Massively Parallel Processors, Morgan Kaufmann Publishers, Burlington, MA, 2010.
6. Frigo, M., Johnson, S.G. The design and implementation of FFTW3. Proc. IEEE 2005, 93(2), 216–31.
7. NVIDIA: Santa Clara, CA, CUDA Programming Guide, http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_3.0.pdf (Accessed March 6, 2010).
8. AMD: Sunnyvale, CA, ATI, www.amd.com/stream (Accessed March 14, 2010).
9. NVIDIA: Santa Clara, CA, CUDA, http://www.nvidia.com/object/cuda_home.html (Accessed March 6, 2010).
10. NVIDIA: Santa Clara, CA, CUFFT Library, http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/CUFFT_Library_2.3.pdf (Accessed March 6, 2010).
11. NVIDIA: Santa Clara, CA, CUBLAS Library 2.0, http://developer.download.nvidia.com/compute/cuda/2_0/docs/CUBLAS_Library_2.0.pdf (Accessed March 6, 2010).
12. Innovative Computing Laboratory, University of Tennessee, Matrix Algebra on GPU and Multicore Architectures, http://icl.cs.utk.edu/magma (Accessed March 6, 2010).
13. Kohn, W., Sham, L. Self-consistent equations including exchange and correlation effects. Phys. Rev. 1965, 140, A1133–8.
14. Parr, R.G., Yang, W. Density-Functional Theory of Atoms and Molecules, Oxford University Press, Oxford, 1989.
15. Jensen, F. In Annual Reports in Computational Chemistry (ed D.C. Spellmeyer), Vol. 1, Elsevier, Amsterdam, 2005, pp. 3–17.
16. Fiolhais, C., Nogueira, F., Marques, M.A.L. A Primer in Density Functional Theory, Lecture Notes in Physics, Springer Verlag, Berlin, 2003.
17. Salek, P., Høs, S., Thøgersen, L., Jørgensen, P., Manninen, P., Olsen, J., Jansík, B. Linear-scaling implementation of molecular electronic self-consistent field theory. J. Chem. Phys. 2007, 126, 114110.
18. Yasuda, K. Two-electron integral evaluation on the graphics processor unit. J. Comput. Chem. 2007, 29(3), 334–42.
19. Dupuis, M., Rys, J., King, H.F. Evaluation of molecular integrals over Gaussian basis functions. J. Chem. Phys. 1976, 65, 111–16.
20. Ufimtsev, I.S., Martínez, T.J. Graphical processing units for quantum chemistry. Comput. Sci. Eng. 2008, 10(6), 26–34.
21. Ufimtsev, I.S., Martínez, T.J. Quantum chemistry on graphical processing units. 1. Strategies for two-electron integral evaluation. J. Chem. Theory Comput. 2008, 4(2), 222–31.

22. McMurchie, L.E., Davidson, E.R. One- and two-electron integrals over Cartesian Gaussian functions. J. Comput. Phys. 1978, 26, 218–31.
23. Ufimtsev, I.S., Martinez, T.J. Quantum chemistry on graphical processing units. 2. Direct self-consistent-field implementation. J. Chem. Theory Comput. 2009, 5(4), 1004–15.
24. Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M., Montgomery, J.A., Jr. General atomic and molecular electronic structure system. J. Comput. Chem. 1993, 14(11), 1347–63.
25. Ufimtsev, I.S., Martinez, T.J. Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics. J. Chem. Theory Comput. 2009, 5(10), 2619–28.
26. 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. J. Chem. Theory Comput. 2010, 6(3), 696–704.
27. Yasuda, K. Accelerating density functional calculations with graphics processing unit. J. Chem. Theory Comput. 2008, 4(8), 1230–6.
28. Perdew, J.P., Chevary, J., Vosko, S., Jackson, K.A., Pederson, M.R., Singh, D., Fiolhais, C. Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation. Phys. Rev. B 1992, 46, 6671–87.
29. ClearSpeed: Bristol, UK, www.clearspeed.com (Accessed March 14, 2010).
30. Brown, P., Woods, C., McIntosh-Smith, S., Manby, F.R. Massively multicore parallelization of Kohn-Sham theory. J. Chem. Theory Comput. 2008, 4(10), 1620–6.
31. Brown, P., Woods, C.J., McIntosh-Smith, S., Manby, F.R., A massively multicore parallelization of the Kohn-Sham energy gradients, J. Comput. Chem. 2010, 31(10), 2008–13.
32. Baerends, E.J., Ellis, D., Roos, P. Self-consistent molecular Hartree-Fock-Slater calculations. I. The computational procedure. Chem. Phys. 1973, 2, 41–51.
33. Dunlap, B.I., Connoly, J.W.D., Sabin, J.R. On some approximations in applications of Xα theory. J. Chem. Phys. 1979, 71, 3396–402.
34. Eichkorn, K., Treutler, O., Öhm, H., Häser, M., Ahlrichs, R. Auxiliary basis sets to approximate Coulomb potentials (Chem. Phys. Lett. 1995, 240, 283) Chem. Phys. Lett. 1995, 242, 652–60.
35. Genovese, L., Ospici, M., Deutsch, T., Méhaut, J.-F., Neelov, A., Goedecker, S. Density functional theory calculation on many-cores hybrid CPU-GPU architectures. J. Chem. Phys. 2009, 131, 34103.
36. Feyereisen, M.W., Fitzgerald, G., Komornicki, A. Use of approximate integrals in ab initio theory. An application in MP2 energy calculations. Chem. Phys. Lett. 1993, 208, 359–63.
37. Weigend, F., Häser, M., Patzelt, H., Ahlrichs, R. RI-MP2: Optimized auxiliary basis sets and demonstration of efficiency. Chem. Phys. Lett. 1998, 294, 143–52.
38. Vogt, L., Olivares-Amaya, R., Kermes, S., Shao, Y., Amador-Bedolla, C., Aspuru-Guzik, A. Accelerating resolution-of-the-identity second-order Møller-Plesset quantum chemistry calculations with graphical processing units. J. Phys. Chem. A 2008, 112(10), 2049–57.
39. Olivares-Amaya, R., Watson, M.A., Edgar, R.G., Vogt, L., Shao, Y., Aspuru-Guzik, A. Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library. J. Chem. Theory Comput. 2010, 6(1), 135–44.
40. SciGPU-GEMM v0.8, http://www.chem-quantum.info/scigpu/?p=61 (Accessed March 6, 2010).
41. Ceperley, D., Alder, B. Quantum Monte Carlo. Science 1986, 231(4738), 555–60.
42. Anderson, A.G., Goddard, W.A., III, Schröder, P. Quantum Monte Carlo on graphical processing units. Comput. Phys. Commun. 2007, 177(3), 298–306.
43. Meredith, J.S., Alvarez, G., Maier, T.A., Schulthess, T.C., Vette, J.S. Accuracy and performance of graphics processors: A quantum Monte Carlo application case study. Parallel Comput. 2009, 35(3), 151–63.