Computer Physics Communications 185 (2014) 2510–2518

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Accelerating atomistic calculations of quantum energy eigenstates on graphic cards

Walter Rodrigues[a], A. Pecchia[b], M. Lopez[a], M. Auf der Maur[a], A. Di Carlo[a,*]

[a] Department of Electronics Engineering, University of Rome Tor Vergata, Via del Politecnico 1, 00133, Rome, Italy
[b] CNR-ISMN, Via Salaria Km 29, 600, 00017 Monterotondo, Rome, Italy

ARTICLE INFO

Article history:
Received 31 December 2013
Received in revised form
18 April 2014
Accepted 22 May 2014
Available online 9 June 2014

Keywords:
Tight-binding
Atomistic simulation
Eigensolver
Lanczos
CUDA
InfiniBand

ABSTRACT

Electronic properties of nanoscale materials require the calculation of eigenvalues and eigenvectors of large matrices. This bottleneck can be overcome by parallel computing techniques or the introduction of faster algorithms. In this paper we report a custom implementation of the Lanczos algorithm with simple restart, optimized for graphical processing units (GPUs). The whole algorithm has been developed using CUDA and runs entirely on the GPU, with a specialized implementation that spares memory and reduces at most machine-to-device data transfers. Furthermore parallel distribution over several GPUs has been attained using the standard message passing interface (MPI). Benchmark calculations performed on a GaN/AlGaN wurtzite quantum dot with up to 600,000 atoms are presented. The empirical tight-binding (ETB) model with an $sp^3d^5s^*$+spin–orbit parametrization has been used to build the system Hamiltonian (H).

1. Introduction

Atomistic calculations of electronic properties of solids play an increasing role in guiding and explaining experimental findings in modern chemistry, material science and semiconductor research. Density functional theory (DFT) has proved to be extremely successful in studying structural properties and electronic states of materials from which formation energies, phase stability and thermodynamic properties can be understood or even predicted. Many particle corrections can be introduced as a perturbation, allowing also the exploration of optical properties. DFT requires the computation of the complete electronic density that can be obtained by a full diagonalization of the system Hamiltonian, leading to a typical $O(N^3)$ scaling. The introduction of localized basis approaches, like atomic and pseudo-atomic orbitals, wavelets or B-spline, thanks to interaction locality, has enabled the construction of massively parallel codes and O(N) computational schemes that have considerably increased the maximum system size, making million atom DFT computations affordable [1–3]. Still, such codes require large high performance computing facilities to run, reducing somehow the accessibility to a wider range of users. Limited computational resources impose restrictions either on the system size or forces one to introduce further approximations in the level of theory. Electronic properties of large-scale systems can be studied at an atomistic level, for instance, using the empirical tight-binding (ETB) method [4,5]. In contrast to DFT where all occupied bands are needed in order to construct the electronic density, in ETB only a few single particle states near the valence and conduction bands are computed. This is justified by the fact that near equilibrium electrons and holes occupy few of such states. ETB employs an implicit basis composed of localized atomic-like orbitals (or Wannier functions) that do not require the computation of overlap and Hamiltonian matrix elements from explicit wave functions. These matrix elements are typically obtained empirically from fits to more accurate calculations or to experiments, but they can also be derived from first-principles expressions. The ETB method is generally less accurate and less transferable than methods based on DFT, but it does provide a good alternative for computing the electronic properties of systems of a larger size. In fact the ETB is the model of choice for atomistic description of the electronic properties of nanostructured devices [4].

* Corresponding author.
E-mail address: aldo.dicarlo@uniroma2.it (A. Di Carlo).

http://dx.doi.org/10.1016/j.cpc.2014.05.028
0010-4655/© 2014 Elsevier B.V. All rights reserved.

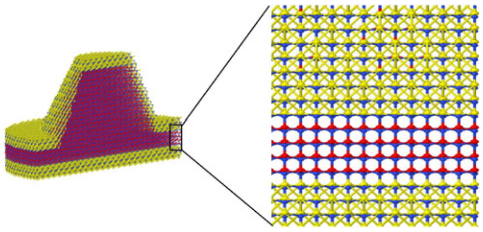

Fig. 1. Conical wurtzite GaN/AlGaN quantum dot (Qdot) with 30% Al. Atomistic description: in yellow Aluminum, in red Gallium.

In this work we apply an ETB model based on an $sp^3d^5s^*$ parametrization [6] to compute eigenpairs of large GaN/AlGaN quantum dots (Qdot), as shown in Fig. 1. Such systems have important applications in modern nitride-based light emitting diodes (LEDs) [7,8]. From the energy states we can compute optical matrix elements and emission spectra of the dot providing a guide for device optimization.

Most eigenvalue solvers have concentrated on computational techniques that accelerate separate components, in particular the matrix–vector multiplication [9], or on new efficient sparse matrix storage formats [10]. Iterative methods based on the Krylov subspace are usually employed to compute few inner eigenstates of large sparse matrices. Among these methods are the original Lanczos algorithm, Arnoldi [11], Krylov–Schur or Jacobi–Davidson [12]. Some of the main standard libraries that include iterative eigensolver routines are ARPACK (ARnoldi PACKage) [13], PRIMME (PReconditioned Iterative MultiMethod Eigensolver) a library based on Jacobi–Davidson algorithm [14], IETL (Iterative Eigensolver Template Library) providing a generic template interface to performance solvers [15] and SLEPc, a scalable library based on the linear algebra package PETSc [16]. All these libraries support single and double precisions, real or complex arithmetic and distributed computations via MPI.

General purpose graphics processing units (GPUs) have become very attractive to computational scientists, thanks to their massive multi-core parallelization, delivering high throughput capabilities also on double-precision arithmetic, to their increased on-board memory and to the efforts made by vendors in facilitating programmability. There is still, however, a limited amount of work specifically dedicated to developing eigensolvers exploiting the new GPU architectures. Moreover, the speedup reported for a similar applications domain is limited to a factor of around 1.5–2.5 with respect to CPU [17,18].

This work has been motivated by the lack of specialized eigensolvers for large-scale computation on GPUs. We concentrate on addressing some basic problems that hinder the development of efficient eigensolver on GPUs: first, the choice of the algorithm itself. We demonstrate how

to overcome the problem of compute vs. communication gap that exist in GPUs and have also established ways to resolve the computational and memory related bottlenecks. Finally we present a multi-GPU implementation that scales with GPUs. Resulting is an eigensolver that accelerates efficiently large-scale tight-binding calculations as illustrated below in more detail.

1.1. Main contributions

We highlight four of our main areas of enhancements related to the design of an efficient GPU-based eigensolver.

- **Identification and tuning of the Lanczos algorithm:** it's rather common to employ methods that have a quadratic convergence, but they are very expensive in terms of memory usage and not well suited for GPUs. We have identified the Lanczos algorithm as a more fitted method for computing few eigenpairs on a GPU framework that can cope with memory limitations of current GPUs and slow GPU–CPU communication.
- **Highly memory-optimized implementation:** we present the technique of splitting the matrix into its real and imaginary parts which required the development of a new CUDA mixed real/complex arithmetics kernel. Performing the multiplication in a splitted fashion resulted in a 40% memory saving without significant loss of performance.
- **Performance enhancement via communication cost reduction:** in order to reduce memory usage and traffic at the cost of extra flops we calculate the eigenvalues and the eigenvectors using minimal information without saving any subspace vectors.
- **Pure multi-GPU execution:** we show how to subdue the slow communication between GPUs by exploiting the matrix sparsity pattern and moreover, the GPU–GPU communication offered by the new GPUDirect technology. The implementation is fully vectorized and scales with GPUs. Scaling from 1 to 2 GPUs is shown in this work.

2. GPU implementation

The GPU architecture allows for the parallel execution of threads performing elementary operations on a large number of processing cores (2496 on a Nvidia Tesla K20c). Although these processing elements are typically much slower than those of a CPU, the massive parallelism and the large memory-to-cache bandwidth may easily outperform current multi-core CPUs [19]. The key element is that multithreading is handled by the hardware with little overhead [20]. This is unlike traditional CPU, where each individual thread is treated as an entity independent of the others, requiring separate resources such as stack memory, and whose creation and management are not cheap [21]. GPUs are ideal for vector operations, such as matrix–vector multiplications, which are the time-consuming kernel operations of iterative linear solvers and eigensolvers.

However, GPU computation suffers from slow host-to-device data transfers that may hinder performance. Naive implementations of the iterative algorithms only exploiting matrix–vector kernels but implementing intensive GPU–CPU data transfers lead to a very limited performance gain, if any. As shown in this work, this limitation can be overcome by keeping the vectors on the GPU and limiting transfer to few scalars for flow control. In practice porting the Lanczos algorithm for best GPU performance requires a substantial code developments.

The on-board memory available on GPUs limits the problem size that can be solved. For our typical TB parametrization based on 20 orbitals per atom, this corresponds to systems of ≈300,000 atoms, when complex double precision is used. In order to reduce memory