Computer Physics Communications 207 (2016) 274–281

Contents lists available at ScienceDirect

Computer Physics Communications

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

Exact diagonalization of quantum lattice models on coprocessors

T. Siro *, A. Harju

Aalto University School of Science, P.O. Box 14100, 00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 20 July 2015
Received in revised form
24 May 2016
Accepted 6 July 2016
Available online 15 July 2016

Keywords:
Tight binding
Hubbard model
Exact diagonalization
GPU
CUDA
MIC
Xeon Phi

ABSTRACT

We implement the Lanczos algorithm on an Intel Xeon Phi coprocessor and compare its performance to a multi-core Intel Xeon CPU and an NVIDIA graphics processor. The Xeon and the Xeon Phi are parallelized with OpenMP and the graphics processor is programmed with CUDA. The performance is evaluated by measuring the execution time of a single step in the Lanczos algorithm. We study two quantum lattice models with different particle numbers, and conclude that for small systems, the multi-core CPU is the fastest platform, while for large systems, the graphics processor is the clear winner, reaching speedups of up to 7.6 compared to the CPU. The Xeon Phi outperforms the CPU with sufficiently large particle number, reaching a speedup of 2.5.

1. Introduction

In recent years, there has been tremendous interest in utilizing coprocessors in scientific computing, including condensed matter physics [1–4]. Most of the work has been done on graphics processing units (GPU), resulting in impressive speedups compared to CPUs in problems that exhibit high data-parallelism and benefit from the high throughput of the GPU. In 2013, a new type of coprocessor emerged in the market, namely the Xeon Phi by chip manufacturer Intel. The Xeon Phi is based on Intel's many integrated core (MIC) architecture, and features around 60 CPU cores that can be easily programmed with existing paradigms, such as OpenMP and MPI. The performance of the Xeon Phi has also already been investigated in some computational physics research areas with mixed results in comparison to GPUs [5–7].

In this work, we apply the Xeon Phi coprocessor to solving the ground state energy of a quantum lattice model by the Lanczos algorithm and compare its performance to a multi-core CPU and a GPU. Previously, the Lanczos algorithm has been implemented on a GPU with speedups of up to around 60 and 100 in single and double precision arithmetic, respectively, in comparison to a single-core CPU program [8].

We examine the tight binding Hamiltonian

$$H = -t \sum_{\langle ij \rangle} \sum_{\sigma=\uparrow,\downarrow} (c^{\dagger}_{i,\sigma} c_{j,\sigma} + h.c), \tag{1}$$

where $\langle ij \rangle$ denotes a sum over neighboring lattice sites, $c^{\dagger}_{i,\sigma}$ and $c_{i,\sigma}$ are the creation and annihilation operators which respectively create and annihilate an electron at site i with spin σ, and $n_{i,\sigma} = c^{\dagger}_{i,\sigma} c_{i,\sigma}$ counts the number of such electrons. The hopping amplitude is denoted by t. The tight-binding model describes free electrons hopping around a lattice, and it gives a crude approximation of the electronic properties of a solid. The model can be made more realistic by adding interactions, such as on-site repulsion, which results in the well-known Hubbard model [9]. In our basis, however, such interaction terms are diagonal, rendering their effect on the computational complexity insignificant when we consider operating with the Hamiltonian on a vector. The results presented in this paper therefore apply to a wide range of different models.

We will solve the lowest eigenvalue, i.e. the ground state energy, of the Hamiltonian numerically with the exact diagonalization (ED) method. This simply means forming the matrix representation of H in a suitable basis and using the Lanczos algorithm to accurately compute the ground state energy. The major advantage of this method is the accuracy of the results, which are essentially exact up to the numerical accuracy of the floating point numbers. The downside is that using the full basis is very costly, since its size

* Corresponding author.
E-mail address: topi.siro@aalto.fi (T. Siro).

http://dx.doi.org/10.1016/j.cpc.2016.07.018
0010-4655/

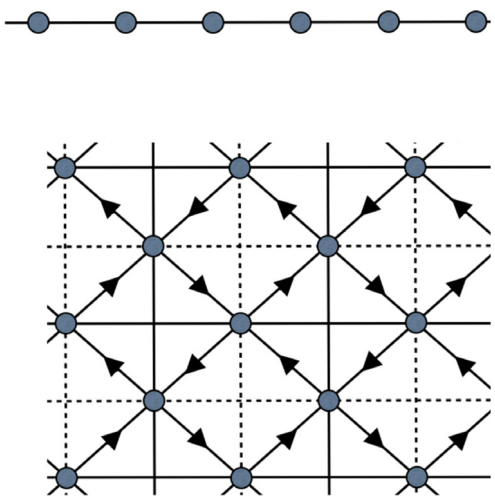

Fig. 1. The two lattice geometries. (Top) A 1D lattice with nearest neighbor hoppings. (Bottom) A checkerboard lattice with complex nearest-neighbor hoppings (the arrows indicate the sign of the complex phase), real next-nearest neighbor hoppings with alternating sign (indicated by the dashed and solid lines) and real third nearest-neighbor hoppings (not drawn for clarity).

scales exponentially with increasing system size and particle number. This means that we are limited to quite small systems. Despite this limitation, the ED method has been successful in many very topical areas of physics, including e.g. the topological properties of condensed matter systems [10–12].

2. Exact diagonalization

2.1. The Hamiltonian

In a lattice with N_s sites with $N_\uparrow$ spin up electrons and $N_\downarrow$ spin down electrons, the dimension of the Hamiltonian is just the number of ways of distributing the electrons into the lattice, taking into account the Pauli exclusion principle that forbids two or more electrons of the same spin from occupying the same site. Thus, the dimension is

$$\dim H = \binom{N_s}{N_\uparrow}\binom{N_s}{N_\downarrow}. \tag{2}$$

The size of the basis grows extremely fast. For example, in the half-filled case where $N_\uparrow = N_\downarrow = N_s/2$, for 12 sites $\dim H = 853776$, for 14 sites $\dim H \approx 11.8 \times 10^6$ and for 16 sites $\dim H \approx 166 \times 10^6$. In addition, the matrices are very sparse, because the number of available hops, and thus the number of nonzero elements in a row, grow only linearly while the size of the matrix grows exponentially.

We study two different lattice geometries, presented in Fig. 1. The first is a simple 1-dimensional lattice with nearest-neighbor hoppings. We use a lattice with 26 and 18 sites for the one and two spin species cases, respectively. The other is a checkerboard lattice introduced in Ref. [13]. It is a widely studied lattice, because with a nearest-neighbor interaction, an analogue to the fractional quantum Hall effect can be observed in the lattice without an external magnetic field [10]. It also contrasts the 1D lattice because it is two-dimensional and has twelve hoppings per site, compared to only two in the 1D lattice. This leads to a much denser hopping Hamiltonian. We use a checkerboard lattice with 30 and 18 sites for the one and two spin species cases, respectively. In all lattices, periodic boundary conditions are always used.

For a detailed description of forming and storing the Hamiltonian, see Ref. [8]. A similar scheme has also been used in Ref. [14].

$$A = \begin{pmatrix} 5 & 1 & 0 & 0 \\ 0 & 2 & 7 & 3 \\ 4 & 0 & 6 & 0 \\ 0 & 9 & 8 & 0 \end{pmatrix}$$

$$\Downarrow$$

$$\text{data} = \begin{pmatrix} 5 & 1 & * \\ 2 & 7 & 3 \\ 4 & 6 & * \\ 9 & 8 & * \end{pmatrix} \quad \text{indices} = \begin{pmatrix} 0 & 1 & * \\ 1 & 2 & 3 \\ 0 & 2 & * \\ 1 & 2 & * \end{pmatrix}$$

$$\Downarrow$$

$$\begin{aligned} \text{data} &= (5, 2, 4, 9, 1, 7, 6, 8, *, 3, *, *) \\ \text{indices} &= (0, 1, 0, 1, 1, 2, 2, 2, *, 3, *, *) \end{aligned}$$

Fig. 2. An example of using the ELL format. It produces two smaller matrices from the initial matrix. In practice, these will be converted to vectors in column-major order for the GPU and row-major order for the CPU and the Xeon Phi. The stars denote padding and they are set to zero.

To summarize, the Hamiltonian can be split into spin up and spin down parts as

$$H = H_\uparrow \otimes I_\downarrow + I_\uparrow \otimes H_\downarrow, \tag{3}$$

where I_σ is the identity operator for electrons with spin σ and $\otimes$ is the tensor product. The basis states for a single spin species, up or down, are represented by integers whose set and unset bits correspond to occupied and unoccupied sites, respectively. Then, the hopping Hamiltonians $H_\uparrow$ and $H_\downarrow$ are computed in the basis and stored in the memory in the ELL sparse matrix format.

The ELL format stores a sparse matrix into two dense matrices that contain the nonzero matrix elements and the corresponding column indices. The width of the matrices is the maximum number of nonzero elements per row in the original matrix. For an example of the ELL sparse matrix format, see Fig. 2. The nonzero density for the matrices we have used ranges from 10^{-3} to 10^{-6}. We use ELL instead of other standard formats, such as CSR, because in H, there is quite little variation in the number of nonzeros per row. This means that we do not have to add a lot of padding zeros into the ELL format matrices. Also, in our tests, we found the performance with CSR to be essentially identical to ELL, so we use the simpler method.

2.2. The Lanczos algorithm

Because of the very fast growth of the Hilbert space dimension as a function of the particle number, fully diagonalizing the Hamiltonian is only possible for rather small systems and with only a few particles. Usually, we are mostly interested in the smallest eigenvalues and states. These can be accurately approximated with iterative algorithms, one of which is the Lanczos algorithm [15].

In the Lanczos algorithm, the Hamiltonian is projected onto an orthogonalized basis in a Krylov subspace, defined by

$$\mathcal{K}_m(f, H) = \text{span}(f, Hf, H^2 f, \ldots, H^{m-1} f), \tag{4}$$

where f is a random starting vector and m is the Krylov space dimension. The result of the Lanczos iteration is a tridiagonal matrix, i.e. one with nonzero elements only on the main diagonal and the first sub- and superdiagonals. The dimension of the resulting matrix is equal to m. As m increases, the lowest (and highest) eigenvalue of the matrix gives an increasingly accurate approximation of the corresponding eigenvalue of H. Importantly, sufficient convergence occurs typically already for $m \approx 100$, even when the Hamiltonian matrix is very large.

In Algorithm 1, we give the pseudocode for the Lanczos algorithm. It generates the so called Lanczos basis, $\{f_1, f_2, \ldots, f_m\}$,