Abstract

We propose an implementation of the GPU algorithm for the inversion of special matrices set. This algorithm takes into account that each matrix in the set is differs from others only by its diagonal elements. The algorithm uses a direct product procedure for the matrix inversion. The ability to use massive parallelization for the calculation of the direct product allows to effectively use GPU calculations which speeds up the solution of this problem. We implement and study the properties of this algorithm for complex valued matrices. Using the GPU algorithm for simulation of the disordered 2D-lattice systems allows to achieve significant speed up in calculations.

Motivation

The problem for inversion of a set of matrices with differing diagonal elements is a typical problem in the numerical simulation of disordered systems. In condensed matter physics, the problem of numerical calculation of the electronic spectrum for disordered systems (e.g. substitutional alloys) encounters this kind type of the operations. These simulations requires large amounts of CPU time, even for simple models. Taking into account that at each system realization only the diagonal matrix elements are changing, the GPU algorithm is proposed for speed up these calculations.

The Model

The model describes a set of a different atoms randomly distributed in a regular lattice. The quantity of interest is the so-called Green function. The Green function for the disordered system is in simplified form given by:

$$ G = \frac{1}{H_0 + V} $$

$H_0$ – is the regular part, $V$ – the diagonal (changing) part, $\langle \rangle$ represents averaging over system realizations.

Mathematical Formalism and Background

Consider the set of matrices: $\hat{M}, \hat{M}^2, \ldots \hat{M}^n$:

$$ \hat{M} = X + \hat{C}, \quad \hat{X}_j = X_j\delta_{ij} = \hat{X}' + \hat{C}, $$

$\hat{C}$ – regular part, diagonal elements are changing in each matrix $\hat{M}^n, \hat{M}^2, \ldots \hat{M}$: $X_j = (X_j + \Delta X_j)\delta_{ij}$, where $\Delta X_j$ – random part.

The task is the inversion of this set:

$$ \hat{Q} = (\hat{M})^{-1}, \hat{Q}' = (\hat{M}')^{-1}, \ldots, \hat{Q}^n = (\hat{M}^n)^{-1} $$

The inversion of the $\hat{M}$ differing at only the $i$-th digaonal element $\hat{X}_i = X_i + \Delta X$ can be performed using this relation:

$$ \hat{Q}_i = Q_0 + \frac{\Delta X}{1 - Q_0 \Delta X} \sum_{j} Q_j Q_0 $$

Denote $L = Q_0 = |L| = 1$ - $i$-th column, $R = Q_0 = |R| = 1$ - $i$-th row of the $\hat{Q}$,

$$ \hat{Q} = \hat{Q} + nL \otimes R \Rightarrow \hat{Q} = \hat{Q} + n|L| \otimes |R| $$

$|L| \otimes |R|$ – direct product of two vectors. This relations is a particular case of the Sherman-Morrison formula, widley used in Monte-Carlo simulations.

Graphical Representation

Flowchart of a typical calculation routine, and schematic view of the GPU algorithm for calculation of the direct product of two vectors. The GPU calculations in main block are performed many times without data transfers between the host and the GPU. This fact allows to increase the efficiency of the GPU algorithm. The dimension of the matrices should be a multiple of 16 (the standard block size) for maximum performance of the GPU algorithm.

Lattice simulations

The results of the simulations for a partially filled lattice. On the left panel a schematic representation of lattice with periodically boundary conditions is shown. On the right panel the result of the simulation: the imaginary part of the averaged Green function in the reciprocal space. Lattice size $L = 32 \times 32$, number of the realizations $N_r \sim 2^{13}$. Some auxiliary procedures, such as the Discrete Fourier Transformation are performed using standard GPU matrix algorithms.

Results

The algorithm benchmarks and system simulations were performed at the self-built box (NVIDIA®-GeForce®-GTX 280, Intel®-Core™-Duo CPU E8400 (Gentoo Linux, NVIDIA®-CUDA™Ver. 3.0), 4Gb RAM). The software implementation consist of two parts: (1) CUDA™-realizations of basic operations for matrices and vectors, including GPU realization of the direct product operation, (2) C++ binding for this CUDA™lib.

![Figure: Time-scales for the inversion of the matrix set using GPU and host calculations. Main panel: time vs. matrix size, inset: time vs. number of realizations (complex valued matrix: complex<float> elements). The GPU data are shown as the approximate reference: there are no any specific optimizations and parallelizations in inversion algorithm.

GPU calculations have an advantage in averaging already for the small set of matrices. In typical real computations the GPU algorithm has a smaller efficiency because of periodical data transfers between host and GPU for different system parameters.

The typical calculations speed gain is about 70 times in comparison with single-core CPU algorithms.

Conclusions

- An algorithm for massive matrix inversion using the GPU is implemented and tested, usage of this implementation allows us to notably speed up numerical simulations of systems with diagonal disorder.
- Matrix sets with constant off-diagonal elements allow a faster inversion procedure by significantly decreasing data transfer instances between the host and the GPU.
- An extension to similar problems (e.g. calculation of the determinant for such matrices sets) of this software implementation is also possible.