Implementation trade-offs of the density matrix renormalization group algorithm on kilo-processor architectures

Csaba Nemes*, Gergely Barcza‡, Zoltán Nagy*†, Örs Legeza‡, and Péter Szolgay*†
*Faculty of Information Technology, Péter Pázmány Catholic University, Budapest, Hungary
Email: nemes.csaba@itk.ppke.hu
†Cellular Sensory and Wave Computing Laboratory, Computer and Research Automation Institute,
Hungarian Academy of Sciences, Budapest, Hungary
‡Strongly Correlated Systems ”Lendület” Research Group, Department of Theoretical Solid State Physics,
Wigner Research Centre for Physics, Hungarian Academy of Sciences, Budapest, Hungary

***Abstract*—Numerical analysis of strongly correlated quantum lattice models has a great importance in quantum physics. The exponentially growing size of the Hilbert space makes these computations difficult, however sophisticated algorithms have been developed to balance the size of the effective Hilbert space and the accuracy of the simulation. One of these methods is the density matrix renormalization group (DMRG) algorithm which has become the leading numerical tool in the study of low dimensional lattice problems of current interest. In the algorithm a high computational problem can be translated to a list of dense matrix operations, which makes it an ideal application to fully utilize the computing power residing in both current multi-core processors and novel kilo-processor architectures.**

I. Introduction

DMRG is a variational numerical approach developed to treat low-dimensional interacting many-body quantum systems effectively [1]–[3]. In fact, it has become an exceptionally successful method to study the low energy physics of arbitrary strongly correlated quantum system which exhibits chain-like entanglement structure [4].

The original DMRG algorithm [1] was introduced in 1992 by Steven R. White and was formulated as a single threaded algorithm. In the past various works have been carried out to accelerate the DMRG algorithm [5]–[8], however, none of them took advantage of recent kilo-processor architectures such as the graphical processing unit (GPU).

II. Model

In order to illustrate the underlying features of the algorithm, we apply it to the so-called spin-$1/2$ Heisenberg model. In the model a magnetic system is simulated on a lattice of interacting *spins*, i.e., a microscopic magnetic moment (spin) is localized at each lattice site j which is described by a quantized, two-valued variable, $\sigma_j \in \{\uparrow,\downarrow\}$, related to the two possible orientation of the spin. Limiting the interactions between neighbouring spins – which is often a good approximation – the Hamiltonian of the model is written as

$$H = \frac{1}{2}\sum_{j=1}^{N-1}\left(S_j^+S_{j+1}^- + S_j^-S_{j+1}^+\right) + \Delta\sum_{j=1}^{N-1} S_j^zS_{j+1}^z \qquad (1)$$

where S^+, S^-, S^z are operators acting on a given site. The effect of these operators is to change the orientation of the spin on a given lattice site or to measure its orientation. The overall behaviour of the system can be tuned via the interaction parameter Δ. The explicit matrix representation of an operator $\mathcal{O}$ acting on site j of a chain with N spins is given as

$$\mathcal{O}_j = \bigotimes_{i=1}^{j-1}\mathbb{I}\otimes\mathcal{O}\otimes\bigotimes_{i=j+1}^{N}\mathbb{I} \qquad (2)$$

where $\mathbb{I}$ is the identity and $\mathcal{O}$ is one of the followings

$$S^+ = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix},\ S^- = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix},\ S^z = \frac{1}{2}\begin{pmatrix}-1 & 0\\ 0 & 1\end{pmatrix}. \qquad (3)$$

The Hamiltonian of N spins acts on the tensor product space of dimension 2^N, $(\mathbb{C}^2)^{\otimes N}$, that is the dimension of the complete Hilbert space grows exponentially as the size of the system increases. From the engineering point of view the main task is to find some of the low-lying eigenvalues and eigenvectors of the Hamiltonian by a diagonalization algorithm. In practice, however, instead of solving the problem for the complete Hilbert space directly, various physical phenomena can be exploited to reduce the complexity of the problem.

III. Symmetries to be exploited

In many systems the Hamilton operator does not change the value of a measurable quantity, i.e., it commutes with the operator connected to that measurable quantity. These operators are called symmetry operators and can be used to cast the Hilbert space to smaller independent subspaces [9]. Consequently, instead of solving a large matrix eigenvalue problem, the eigenvalue spectrum can be determined by solving several smaller problems. In the presented model, the total spin projection, $S_z = \sum_{j=1}^{N} S_j^z$, is such a symmetry operator.

A given symmetry operator shares the same eigenvectors of the Hamiltonian, thus the eigenstates of the Hamiltonian can be labelled by the eigenvalues of the symmetry operator (*quantum number*, Q), and the Hilbert space can be decomposed into subspaces (*sectors*) spanned by the eigenvectors of each quantum number value [10]. Introducing a quantum number based representation, the sparse operators (Eq. 2) can be decomposed to a set of smaller but dense matrices, furthermore the Hamiltonian operator (Eq. 1) becomes blockdiagonal.

IV. Algorithm

The DMRG approach has two phases, in the *infinite-lattice algorithm* the approximated Hilbert space of a finite system of N interacting spins is built up iteratively, while in the *finite-lattice algorithm* similar optimization steps are carried out, keeping the size of the problem fixed, in order to increase the accuracy of the computed results. As optimization steps in both cases are similar, for sake of simplicity, we consider only the infinite-lattice algorithm. The detailed description of the algorithm can be found in the original work [1] and various reviews [2], [3], here only the key steps of an iteration of the infinite-lattice algorithm are summarized in Pseudocode 1 providing the basis of our analysis.

Algorithm 1 One iteration of the infinite-lattice algorithm

```
Load a left and a right block.
Form the superblock configuration based on the symmetries.
Compute the lowest eigenstate of the superblock Hamilton $H_{SB}$. (Davidson method)
for each block do
    Construct the density matrix for the given block from the lowest eigenstate.
    Compute the eigenvalues of the density matrix. (Lanczos method)
    Renormalize the basis of the block while keeping states with high eigenvalues.
end for
```

In the two-site DMRG procedure four subsystems (left block describing l sites, 1 site, 1 site, right block describing r sites) compose the finite system of $N = (l+2+r)$ sites called *superblock*. The sites contained in each block are described maximally by m, optimally chosen states, which representation can be significantly smaller then the exactly required 2^l basis. Meanwhile, the central sites of the superblock are represented exactly by 2-2 states, so the size of the superblock Hilbert space is $4m^2$. Considering, however, the projection symmetry mentioned above, the problem can be restricted to a subspace of the superblock corresponding to a particular Q value.

The infinite-lattice algorithm starts with the four site configuration, where both blocks contain a single spin. In each iteration step both blocks are enlarged by a single site, making the complete system increase by two, until the desired system size, N, is reached. In each iteration of the DMRG algorithm, the lowest-lying eigenvector of the corresponding superblock Hamiltonian (H_{SB}) is obtained by the iterative Davidson algorithm.

The most time-consuming part of a full iteration step is the Davidson routine carrying out the matrix-vector multiplication operation ($X' = H_{SB}X$). Instead of constructing and storing the enormous H_{SB} matrix of size $\mathcal{O}\left(m^4\right)$ explicitly, it is computationally favourable to obtain the projected vector X' directly via the matrices of size $\mathcal{O}\left(m^2\right)$ composing H_{SB}.

The H_{SB} can be expressed by the operators of the 4 subsystems (so called *l*-1-1-*r strategy)* and the computation of the projected vector X' can be formulated as 4-indexed tensor multiplications. However, in the paper the simpler *LR strategy* is implemented, where the H_{SB} is expressed by operators $A_\alpha^{(L)}$ and $B_\alpha^{(R)}$, defined on blocks enlarged by the neighbouring site, i.e., $L = l + 1$, $R = r + 1$, as

$$H_{SB} = \sum_\alpha A_\alpha^{(L)} \otimes B_\alpha^{(R)} \ , \tag{4}$$

where the index α iterates over the distinct operator combinations required to construct the superblock Hamiltonian. Furthermore, exploiting Kronecker multiplication properties, the projected vector X' can be computed by matrix-matrix multiplications as

$$\tilde{X}' = \sum_\alpha A_\alpha^{(L)} \tilde{X} B_\alpha^{(R)T} \ , \tag{5}$$

where the vector X of dimension $(mq)^2$ is reshaped to the matrix $\tilde{X}$ dimension of ($mq \times mq$).

In the practical implementation Equation 5 is decomposed to even smaller matrix operations of the same type as the operators are already decomposed according to quantum numbers. This means that instead of a sparse matrix $A^{(L)}$ dense matrices $A^{(L)}_{q_i \to q_j}$ are stored representing how $A^{(L)}$ transforms the subspace (sector) corresponding to q_i to the one corresponding to q_j. For an example $A^{(L)}_{q_i \to q_j}$ and $B^{(R)}_{q_k \to q_l}$ projects only the ik segment of a vector X to the jl segment:

$$\tilde{X}'_{jl} = A^{(L)}_{q_i \to q_j} \tilde{X}_{ik} B^{(R)T}_{q_k \to q_l} \tag{6}$$

where $\tilde{X}_{ik}$ again indicates the reshaped the ik segment of the X vector.

In the rest of the iteration the density matrices are constructed from the lowest eigenstate and used to optimally truncate the basis of the enlarged block ($m \ll 2^L$) in order to keep the problem size manageable.

V. Acceleration strategies

A. Acceleration on CPU

The DMRG implementation shall use as much parallelism as possible to exploit the enormous performance available in nowadays multi-core CPUs. Inherent parallelism can be observed at two level of the algorithm. First, at low level, all the matrix and vector operations can be accelerated in a multi-core environment, secondly at the level of the projection computation the $(AX)B^T$ operations in Equation 5 can be computed independently.

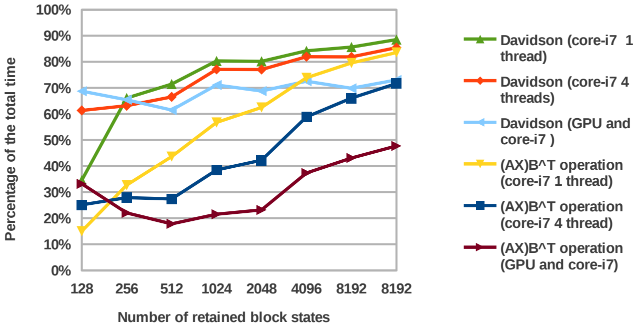

Fig. 1. Runtime of the Davidson algorithm and $(AX)B^T$ operations compared to the total time of a full iteration step as the number of the retained block states increases. In case of CPU all the algebraic operations are accelerated, while in GPU only the $(AX)B^T$ operations are accelerated and the rest tends to be slow.

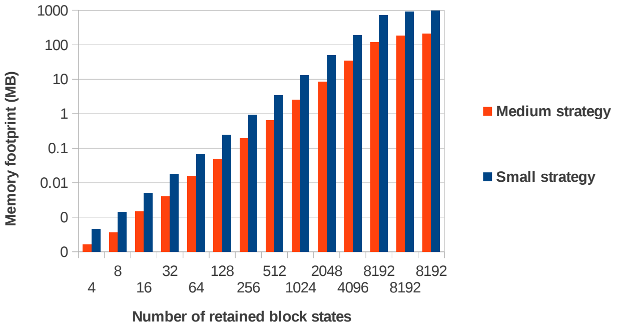

Fig. 2. Memory footprint on GPU in case of two strategies. As the number of kept states doubles, the memory footprint on GPU approximately quadruples. It is obvious the number of sustained states will be affect the selected strategy and thus the performace of the GPU accelerator.

The first is accomplished by using the Basic Linear Algebra Subroutine (BLAS) interface and the Intel MKL Library [11] for algebraic operations including operator contractions, inner operations of both Davidson and Lanczos algorithms [12] and operator transformations. The second parallelism is not implemented yet in the CPU reference code, however, this affects only the small-scale performance. As shown in Figure 3 the CPU performance reaches its maximum ($\sim$ 85 Gflops) in case of $(AX)B^T$ operation at a relatively small matrix size.

As the number of the retained block states increases the Davidson algorithm and the $(AX)B^T$ operations become the most dominant part of the computation (see Figure 1). In case of single threaded CPU implementation the computation time of the independent $(AX)B^T$ operations reaches almost 90% of the total time, therefore, the benefit of the acceleration of these operations on kilo-processors architectures is self-evident.

B. *Acceleration on GPU*

The acceleration of the $(AX)B^T$ operations in GPU is based on the observation that A and B matrices are already available before the Davidson algorithm starts and do not change during the Davidson iterations. The projection operation ($H_{SB}X$) is described by a list of *operation records* in which each record contains the necessary informations to compute an operation like Equation 6. An operation record stores the information from which segment (*input*) of X to which segment (*output*) of X' the operation transforms.

Algorithm 2 Host side algorithm to handle the operation records

```
Partition operation records between CPU and GPU.
Selects scheduling strategy for the operations to be computed on GPU.
Apply scheduling strategy.
```

The host side algorithm to handle the operation records is summarized in Pseudocode 2. First operation records are partitioned between the CPU and the GPU in the ratio of the performance capability of the two architectures. Currently all the operating records corresponding to the same output shall be computed on the same architecture, however, later a more sophisticated partitioning can be implemented. As we already discussed and shown in Figure 3, CPU has a significant performance capability which shall also be exploited in combined CPU+GPU design.

The DMRG algorithm is implemented such a way that three different strategies for scheduling of the competitive $(AX)B^T$ operations can be tested. The first strategy (*small*) is designed for small problem size, when all A, B, X, X' matrices and temporary matrices T for storing intermediate results can be held in the GPU memory. The second strategy (*medium*) is designed for medium-sized problems, where all A, B and X' matrices can be stored in GPU memory. In case of extra-sized problems a third strategy (*large*) can be designed in which only those A and B matrices are loaded which are explicitly needed for the given operation record.

The proper strategy shall be automatically selected based on the memory requirements of the operation records, however, currently only the second strategy is implemented. The memory footprint of the matrices in case of different strategies are shown in Figure 2. In the demonstrated experiment the second strategy was adequate as the GPU card has a memory of 1GB.

The $(AX)B^T$ operations are implemented using the cuBLAS libary [13], which is a BLAS implementation dedicated for Nvidia GPUs. In the demonstrated strategies two important features of the GPUs are exploited, which are provided via the CUDA driver [14] and also accessible through the cuBLAS library. The first feature is that multiple CUDA kernels can be executed simultaneously on the GPU, while the second feature is that memory I/O operations can be executed in the background. From the aspect of programming both feature can be accessed via the CUDA streams. Streams are sequence of operations that execute in issue-order, but operations in different streams may run concurrently or interleaved.

In the small strategy we have enough GPU memory to execute several $(AX)B^T$ simultaneously. We can create one stream for each output, and operations corresponding to a

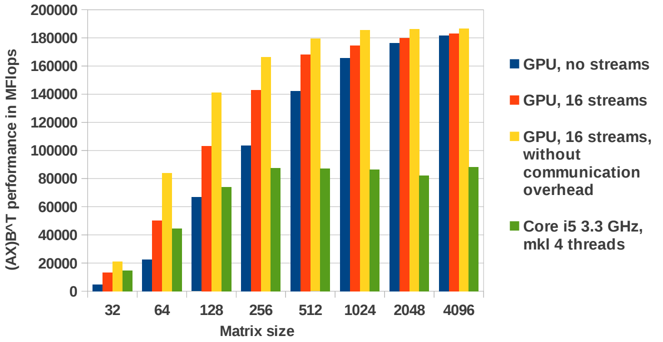

Fig. 3. $(AX)B^T$ performance in MFlops. NVidia GTX 570 is compared to Intel Core-i5 3.3GHz.

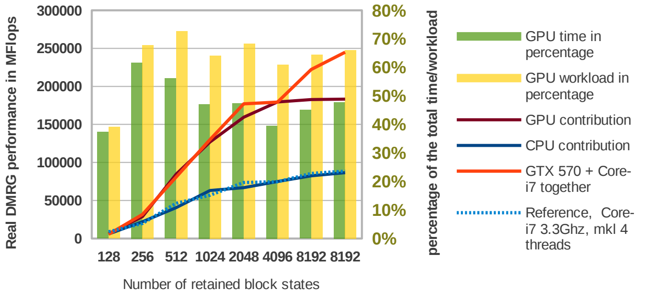

Fig. 4. The performance of the projection operation of the DMRG algorithm. If the workload is properly distributed 244 GFlops can be reached which is a 2.7 speed-up compared to the CPU only reference code (88 GFlops).

given output are assigned to the same stream to avoid interference. Practically one sufficiently large temporary matrix is allocated for each stream to store the temporary result of AX. As shown in Figure 3 when the matrices are small several operations shall be executed concurrently to keep all the CUDA cores busy resulting in a higher performance. Operations on large matrices provide enough work for each CUDA core to reach the maximum double performance without streams.

In the medium strategy only A and B matrices and the outputs X' can be stored in the GPU memory. In this scenario only one working stream can be implemented, however, the size of matrices are in the range where the multiple streams have no advantages. To hide the latency of loading of inputs X a dedicate I/O stream is used to implement a double buffered reading. Operation records are sorted based on inputs and while the operations corresponding to the same input are processed the next input can be loaded in the background.

VI. Implementation Results

The presented GPU+CPU combined implementation is compared to the CPU only reference code in a PC equipped with an Intel Core-i7 2700 3.4 GHz processors and an NVidia GTX 570 GPU in Figure 4. The quality of the partitioning of workload highly affects the performance gain reached by the combined version, therefore, more attention shall be paid to it later.

GPU and CPU contributions to the overall performance can be compared to the maximum performance achievable via the $(AX)B^T$ operation on the given architectures (see Figure 3). In the range of 2048..8096 problem size the medium strategy looks promising and both architectures operate with an acceptable performance. If the workload is properly distributed 244 GFlops can be reached resulting in a 2.7 speed-up compared to the performance of the reference code (88 GFlops).

VII. Conclusion

The most computationally demanding part of the DMRG algorithm is a set of dense matrix operations of type $(AX)B^T$ which can be executed independently. The dense matrix operation is an ideal application to exploit the performance capabilities of both CPU and GPU, in the later case even reaching the maximum performance what CUDA cores can give. In our case the performance of core-i7 CPU is not negligible compared to the GPU, therefore, a combined CPU+GPU application has been designed where the workload can be shared to reach a 2.7 speed-up even with a midrange GPU. As a future work a 6 times more powerful GPU (K20) will also be tested.

References

[1] S. R. White, “Density matrix formulation for quantum renormalization groups,” *Phys. Rev. Lett.*, vol. 69, pp. 2863–2866, Nov 1992.

[2] R. M. Noack and S. R. Manmana, “Diagonalization and Numerical Renormalization-Group-Based Methods for Interacting Quantum Systems,” *AIP Conf. Proc.*, vol. 789, pp. 93–163, October 2004.

[3] U. Schollwöck, “The density-matrix renormalization group,” *Rev. Mod. Phys.*, vol. 77, pp. 259–315, Apr 2005.

[4] O. Legeza, R. Noack, J. Sólyom, and L. Tincani, “Applications of quantum information in the density-matrix renormalization group,” in *Computational Many-Particle Physics*, ser. Lecture Notes in Physics. Berlin Heidelberg: Springer-Verlag, 2008, vol. 739.

[5] G. K.-L. Chan, “An algorithm for large scale density matrix renormalization group calculations,” *J. Chem. Phys.*, vol. 120, Dec 2004.

[6] G. Hager, E. Jeckelmann, H. Fehske, and G. Wellein, “Parallelization strategies for density matrix renormalization group algorithms on shared-memory systems,” *Journal of Computational Physics*, vol. 194, no. 2, pp. 795 – 808, 2004.

[7] Y. Kurashige and T. Yanai, “High-performance ab initio density matrix renormalization group method: Applicability to large-scale multireference problems for metal compounds,” *J. Chem. Phys.*, vol. 130, June 2009.

[8] S. Yamada, M. Okumura, and M. Machida, “High performance computing for eigenvalue solver in density-matrix renormalization group method: Parallelization of the hamiltonian matrix-vector multiplication,” in *High Performance Computing for Computational Science - VECPAR 2008*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5336, pp. 39–45.

[9] A. I. Tóth, C. P. Moca, O. Legeza, and G. Zaránd, “Density matrix numerical renormalization group for non-abelian symmetries,” *Phys. Rev. B*, vol. 78, p. 245109, Dec 2008.

[10] J. F. Cornwell, *Group Theory in Physics, An Introduction*. Academic Press, Jul. 1997.

[11] Intel, “Math kernel library 11.0,” http://http://software.intel.com/en-us/intel-mkl, 2013.

[12] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*. Manchester: Manchester University Press, 1992.

[13] “CUBLAS library 5.0,” https://developer.nvidia.com/cublas, NVIDIA Corp, 2013.

[14] “CUDA library 5.0,” http://www.nvidia.com/object/cuda_home_new.html, NVIDIA Corp, 2013.