Computer Physics Communications 181 (2010) 1346–1351

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Improved parallelization techniques for the density matrix renormalization group

Julián Rincón *, D.J. García, K. Hallberg

Centro Atómico Bariloche and Instituto Balseiro, Comisión Nacional de Energía Atómica, CONICET, 8400 Bariloche, Argentina

ARTICLE INFO

Article history:
Received 29 September 2009
Received in revised form 27 March 2010
Accepted 30 March 2010
Available online 2 April 2010

Keywords:
Density-matrix renormalization
Distributed programming
MPI

ABSTRACT

A distributed-memory parallelization strategy for the density matrix renormalization group is proposed for cases where correlation functions are required. This new strategy has substantial improvements with respect to previous works. A scalability analysis shows an overall serial fraction of 9.4% and an efficiency of around 60% considering up to eight nodes. Sources of possible parallel slowdown are pointed out and solutions to circumvent these issues are brought forward in order to achieve a better performance.

1. Introduction

The impact of numerical methods in the study of phenomena which are hardly understood by means of analytical machinery has been decisive. Hence, the current algorithms ought to be constantly assessed regarding the emergence of new concepts and the increasing computing technology. Nowadays one of the most successful algorithms dealing with one-dimensional interacting systems is the so-called Density Matrix Renormalization Group (DMRG) [1]. Although this method is not, strictly speaking, a renormalization procedure, the key idea is the decimation of the Hilbert space by appealing to the concept of the reduced density matrix. This fundamental concept has permitted implementing the DMRG to an extensive variety of systems and physical problems such as small grain physics, classical 2D systems, nuclear physics, quantum information, quantum chemistry, bosonic and fermionic degrees of freedom, and spin systems, together with finite temperature and non-equilibrium problems [2,3].

In most of the interesting physical situations, one has to deal with very large systems in order to prevent, for instance, finite-size effects. This fact leads unavoidably to exhaust single-machine resources. Additionally, as the dimension of the problem increases, the computational costs become more demanding. Bearing this in mind, it seems natural to request for a distributed kind of calculation. Earlier proposals consisted on shared-memory approaches [4] for the DMRG: this method was based on the multithreaded API (Application Programming Interface), namely, OpenMP [5]. Distributed-memory versions of DMRG have been recently proposed in several contexts [6–9]. For DMRG calculations in quantum chemistry very powerful parallel algorithms have been proposed with two basic approaches: (i) the clever distribution of the local, doubly, and triply contracted orbital operators with an almost linear speedup [6], or (ii) the dynamical scheduling of the sub-blocks of the orbital operators labeled by their corresponding quantum numbers [7]. Concerning strongly correlated systems, there have been a few solutions to handle two-dimensional geometries by coding a parallelization that converts the superblock vectors into distributed matrices [8], or a generic version of a one-dimensional DMRG including a parallelization over symmetry-related matrix blocks [9].

The main idea behind these methods was to parallelize the central operation of a ground-state DMRG simulation: the matrix–vector multiplication in the diagonalization of the superblock Hamiltonian. However, the scheme does not take into account calculations of measurements such as expectation values, multiple-point correlation functions, and structure factors, for which the most time-consuming part of the algorithm is the huge amount of matrix–matrix multiplications (i.e. density-matrix rotations) of the operators one is interested in. In addition, the shared-memory scheme would already show scalability problems in a large-scale computation including the calculation of such physical quantities.

In this work, in addition to recoding the ground-state DMRG in the well-known passing message standard MPI [10] (henceforth *regular parallelization*), we propose an improved strategy that takes into account the heavy rotations associated to the calculation of the correlation functions; this policy is also implemented in MPI allowing us to perform genuine high-performance simulations [6]. Two approaches to deal with these rotations are proposed. The first strategy is based on a pool of tasks in which there is a master node distributing queues to the rest of the slaves. The second application performs a block-fashion single distribution considering all nodes with an equal amount of work, hereafter the *uniform-*

* Corresponding author. Tel.:+54 2944 445378.
E-mail address: julian.jimenez@cab.cnea.gov.ar (J. Rincón).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2010.03.018

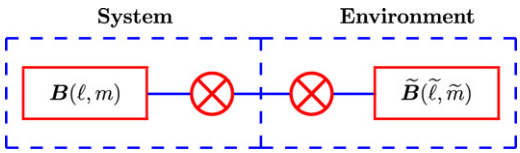

Fig. 1. DMRG block configuration. The superblock is formed with two blocks ($\boldsymbol{B}$ and $\tilde{\boldsymbol{B}}$) and two (exact) sites. Added sites $\boldsymbol{a}$ and $\tilde{\boldsymbol{a}}$ are shown as circles. Dashed lines represent *system* and *environment* blocks. The tilde on the right block means that no reflection symmetry has been assumed.

matrix distribution (UMD) strategy. The latter is easier to implement and more efficient than the former. We obtain similar results for the speedup and performance to previously reported ground-state DMRG simulations with OpenMP. The chosen benchmark was the one-dimensional Hubbard model [11].

In the forthcoming sections the DMRG algorithm will be briefly described, then the usual and new parallelization strategies will be presented and speedup/performance results are analyzed. Thereupon, an application test on the Hubbard model is done to estimate the runtime improvement due to the parallelization ideas of the previous sections, and we finally summarize significant concepts.

2. The DMRG algorithm

This variational, non-perturbative and highly accurate method [2] was developed as an attempt to solve the low-lying energy properties of many-body models that techniques such as exact and Lanczos diagonalization [12], numerical renormalization group (NRG) [13] or other analytical tools could not be able to deal with; moreover this method does not have the sign problem that emerges in Monte Carlo techniques [14]. It can be considered as an improved version of Wilson's NRG for which the states kept during the decimation procedure are no longer selected regarding their energy but instead, they are chosen by means of the density matrix, which naturally gives the most relevant states to be kept (with respect to, e.g. the lowest-lying eigenstate of the whole system).

The standard configuration used in the DMRG algorithm is shown in Fig. 1. We assume the following notation: $\boldsymbol{B}(\ell, m)$ a block composed of ℓ sites with a Hilbert space of dimension m and $\boldsymbol{a}(\ell', m')$ a small added block (usually a single site, e.g., for the Hubbard model case: $\ell' = 1$ and $m' = 4$). Therefore, the *superblock* is formed by the union of two blocks and two sites as shown in Fig. 1. This superblock is built up of two main parts: the *system* and the *environment* composed by a block–site each. $\boldsymbol{B}(\ell, m)$ is a vector space with a completeness relation close to but not equal to $\mathbf{1}$ due to the decimation process.[1] On the contrary, the subspace $\boldsymbol{a}$ is always complete.

The main goal is typically the lowest-energy (ground) state of the superblock Hamiltonian H which can be written as

$$|\psi_0\rangle = \sum_i \sum_j \psi_{0,ij} |i\rangle \otimes |j\rangle, \tag{1}$$

where $\{|i\rangle\}$ and $\{|j\rangle\}$ stand for the orthonormal basis for the *system* and the *environment* respectively and $\psi_{0,ij} = \langle i \otimes j | \psi_0 \rangle$. A truncation procedure should be now established in order to get manageable Hilbert spaces. To this end, DMRG resorts to the reduced density matrix of the *system*:

$$\rho_{ii'} = \sum_j \psi_{0,ij} \psi^*_{0,i'j}. \tag{2}$$

[1] For the reader not familiar with DMRG, blocks and sites can be thought of as vector spaces on which there are certain conditions for well-defined states and operators.

This matrix possesses non-negative eigenvalues w_α with eigenvectors $|w_\alpha\rangle$ ($\rho|w_\alpha\rangle = w_\alpha|w_\alpha\rangle$). It can be shown [3] that these eigenvalues are proportional to the probability of the *system* being in the state $|w_\alpha\rangle$. Selecting the corresponding eigenstates which have the largest probabilities w_α, we can set a cutoff such that we have a very efficient decimation formula. This error source can be quantitative described by defining the *truncation error*

$$\epsilon_\rho = 1 - \sum_\alpha^m w_\alpha, \tag{3}$$

where m is the cutoff, a truncation number selected often by hand. It can be shown [2,3] that the error in the ground state goes as $\| |\bar{\psi}_0\rangle - |\psi_0\rangle \|^2 = \epsilon_\rho$ where $|\bar{\psi}_0\rangle$ is the DMRG approximation to the exact ground state. A similar bound can be found for the expectation values. It is also shown that the energies obtained with DMRG will be upper bounds on the exact eigenvalues. From Eq. (3) it is evident that the more states are kept the higher the accuracy of the calculated energies and observables will be. Another (generally smaller) source of error in $|\psi_0\rangle$ is due to the iterative method used to diagonalize the superblock Hamiltonian. As a consequence of the Hilbert space truncation there is an *environmental error* which has to do with the fact that the bath coupled to the *system* is not exact. The environmental error can be reduced by implementing the so-called finite system algorithm.

The arrangement shown in Fig. 1 is usually used in two ways: on one hand, the *infinite system algorithm* in which the superblock size is grown by adding two new sites in the middle of the chain at each iteration step. And on the other hand, the *finite system algorithm* is designed to calculate highly accurate properties of the superblock at a given lattice length. It consists on moving back and forward (sweeping) the division between *system* and *environment* (it can be thought of as a thermalization of the *system* and *environment* blocks).

All these steps can be summarized in the following way:

1. Start with left and right blocks as exact single sites.
2. Diagonalize the superblock Hamiltonian H defined on $[\boldsymbol{B}(\ell, m)\boldsymbol{a}\tilde{\boldsymbol{a}}\tilde{\boldsymbol{B}}(\tilde{\ell}, \tilde{m})]$ to obtain $|\psi_0\rangle$.
3. Build up all of the block operators related to H and measurements defined on $[\boldsymbol{Ba}] \doteq [\boldsymbol{B} \oplus \boldsymbol{a}]$.
4. Define and diagonalize ρ in the *system*. Find the rotation matrix $\mathcal{R} = (|w_1\rangle|w_2\rangle \cdots |w_m\rangle)^T$ formed from the m largest eigenvalues w_α of ρ.
5. Perform the decimation and rotation step $[\boldsymbol{B}] \longleftarrow \mathcal{R}[\boldsymbol{Ba}]\mathcal{R}^+$ for the operators defined in step 3.
   Go to step 2.

When the desired system size has been achieved, measurements of the relevant quantities such as structure factors, spin and charge gaps, binding energies, etc. can be performed.

Since our main concern is the computation of n-point correlation functions for several operators Z, we have to provide a form for such matrices. This type of simulation can be included in the standard algorithm just managing those Z operators in the same way as the superblock Hamiltonian operators are handled, that is, by doing the transformations of blocking $Z_{[\boldsymbol{Ba}]} \longleftarrow Z_{[\boldsymbol{B} \oplus \boldsymbol{a}]}$ and then the rotation and the decimation step $Z_{[\boldsymbol{B}]} \longleftarrow \mathcal{R} Z_{[\boldsymbol{Ba}]} \mathcal{R}^+$. All of the operators Z are managed as block matrices instead of as block–site matrices reducing the consumed computational resources and saving time on I/O operations.

2.1. Benchmark

We have tested the parallel algorithm with a simulation of the one-dimensional quarter-filled Hubbard model [11]. The Hamiltonian of the model reads:

$$H = -t\sum_{i,\sigma}\left(c^{+}_{i+1\sigma}c_{i\sigma} + c^{+}_{i\sigma}c_{i+1\sigma}\right) + \frac{U}{2}\sum_{i,\sigma} c^{+}_{i\sigma}c_{i\sigma}c^{+}_{i\bar{\sigma}}c_{i\bar{\sigma}}, \tag{4}$$

where $c_{i\sigma}$ ($c^{+}_{i\sigma}$) denotes an electron annihilation (creation) operator on site i with spin $\sigma = (\uparrow, \downarrow)$. Here, $c_{i\sigma}$ is an $m \times m$ matrix. Regarding storage effects, σ implies two different matrices for $c_{i\sigma}$ for each site i. t and U are parameters standing for electron hopping and on-site electron repulsion respectively.

The charge $N(q)$ and spin $S^z(q)$ structure factors

$$N(q) = \frac{1}{L}\sum_{k,j} e^{iq(k-j)}\left\langle (n_k - n)(n_j - n)\right\rangle,$$

$$S^z(q) = \frac{1}{L}\sum_{k,j} e^{iq(k-j)}\left\langle S^z_k S^z_j\right\rangle \tag{5}$$

were calculated, the number operator is $n_{i\sigma} = c^{+}_{i\sigma}c_{i\sigma}$, $n_i = n_{i\uparrow} + n_{i\downarrow}$, and n is the charge expectation value. As it can be seen, obtaining these two quantities requires the calculation of the expectation values $\langle n_i\rangle$ and all of the charge–charge $\langle n_i n_j\rangle$ and spin–spin $\langle S^z_i S^z_j\rangle$ correlation functions.

3. Parallelization

There are two main architecture paradigms in parallel computing: systems with a single address space called *shared-memory systems* allowing multiple processors to access the same memory location (data) and *distributed-memory systems* in which each processor has its own address space and therefore its own data structure. Both paradigms can be successfully applied to the DMRG method [4,6]. Earlier distribution strategies worked well on a shared-memory system methodology; nevertheless, this type of architecture eludes a massively parallel approach. Consequently, a distributed-memory policy should be developed in order to get a coarse-grain scheme reaching larger lengths and more states per block using modest computational resources. Here, in addition of putting forward a new parallelization scheme, we have changed the shared-memory (OpenMP) approach to a standard message passing API (MPI) [10].

As we will show below very similar results are obtained to the OpenMP case with the possibility of improving scalability properties. This distributed approach has the advantage of avoiding collisions (present on MP algorithms) at the presumable cost of using more resources and larger communications. The calculations presented in this work were performed using a cluster with Intel® Xeon 2.50 GHz CPU cores (with a memory of 1 GB per node) arranged either as a double quad-core system or as single cores in a star topology network with a nominal bandwidth of 900 Mb/s.

Let us now briefly summarize the analytical apparatus needed to study the speed of a high-performance realization [15]. The *speedup* S_p indicates how much faster a parallel code on a p-node process is with respect to the sequential analogue. S_p is explicitly defined as the fraction

$$S_p = \frac{T_1}{T_p}, \tag{6}$$

where T_1 and T_p are the wall-clock times of the simulation with 1 and p processors respectively. The ideal speedup should scale linearly with p, that is, $S_p^{\text{ideal}} = p$. Another quantity of interest which illustrates how much the algorithm is exploiting a single processor is the *efficiency* which reads

$$E_p = \frac{S_p}{p}. \tag{7}$$

In the simplest model, the sequential time of a program (normalized to 1) can be split into a *serial fraction* Σ and a *parallel fraction* $1 - \Sigma$. With a finite number of nodes p, the parallel fraction gets reduced by $(1 - \Sigma)/p$; based on these considerations we obtain *Amdahl's law* [16] for the relative speedup

$$S_A(p) = \left(\Sigma + \frac{1 - \Sigma}{p}\right)^{-1}, \tag{8}$$

thus, the maximum speedup achievable (i.e. with $p \to \infty$) would be $S_A \to 1/\Sigma$. This amount gives us a rough idea of the expected efficiency in a distributed implementation.[2]

3.1. Regular parallelization: ground-state DMRG

It is well known that the most time-consuming part in the ground state DMRG is obtaining the lowest eigenvalue of the superblock Hamiltonian H by means of an iterative procedure (such as Lanczos [12] or Davidson [17] algorithms). Since H is actually a sum of terms involving left (*system* formed by $\boldsymbol{B} \oplus \boldsymbol{a}$) and right (*environment* formed by $\tilde{\boldsymbol{a}} \oplus \tilde{\boldsymbol{B}}$) matrix products, we can readily write

$$H = \sum_{\lambda} O^{\lambda}_{\boldsymbol{B}} \otimes O^{\lambda}_{\boldsymbol{a}} \otimes O^{\lambda}_{\tilde{\boldsymbol{a}}} \otimes O^{\lambda}_{\tilde{\boldsymbol{B}}}, \tag{9}$$

where $O^{\lambda}_{\boldsymbol{X}}$ represents a generic operator defined on any of the blocks ($\boldsymbol{X} = \boldsymbol{B}, \boldsymbol{a}, \tilde{\boldsymbol{a}}, \tilde{\boldsymbol{B}}$) and λ corresponds to each of the terms in Eq. (4). Typical terms are for instance, the hopping term between the left block and left site: $c^{+}_{\boldsymbol{B}}c_{\boldsymbol{a}} \equiv c^{+}_{\boldsymbol{B}} \otimes c_{\boldsymbol{a}} \otimes I_{\tilde{\boldsymbol{a}}} \otimes I_{\tilde{\boldsymbol{B}}}$ or the right block Hamiltonian: $H_{\tilde{\boldsymbol{B}}} \equiv I_{\boldsymbol{B}} \otimes I_{\boldsymbol{a}} \otimes I_{\tilde{\boldsymbol{a}}} \otimes H_{\tilde{\boldsymbol{B}}}$ which should contain all of the H terms for the sites belonging to $\tilde{\boldsymbol{B}}$. $I_{\boldsymbol{X}}$ stands for the identity on the space $\boldsymbol{X}$.

If the implementation incorporates symmetries, such as particle number or total magnetization, then H takes the form

$$H = \sum_{\lambda}\sum_{\theta} O^{\lambda}_{\boldsymbol{B}}(\theta^{\boldsymbol{B}}) \otimes O^{\lambda}_{\boldsymbol{a}}(\theta^{\boldsymbol{a}}) \otimes O^{\lambda}_{\tilde{\boldsymbol{a}}}(\theta^{\tilde{\boldsymbol{a}}}) \otimes O^{\lambda}_{\tilde{\boldsymbol{B}}}(\theta^{\tilde{\boldsymbol{B}}}), \tag{10}$$

explicitly showing that the operators $O^{\lambda}_{\boldsymbol{X}}(\theta^{\boldsymbol{X}})$ are labeled by their quantum numbers. The value $\theta^{\boldsymbol{X}}$ is a symmetry index of the $\boldsymbol{X}$ block, and θ is an index running over the superblock basis formed by the configurations with the quantum number $\theta^{\boldsymbol{B}} + \theta^{\boldsymbol{a}} + \theta^{\tilde{\boldsymbol{a}}} + \theta^{\tilde{\boldsymbol{B}}}$ fixed. Using symmetries helps to minimize the size of nested loops. Usually H is a very large matrix (e.g. with dimension $\mathcal{M} \sim 10^4$–10^6), thus it is never explicitly constructed but rather consists of multiplication rules. This means that given a vector $|b\rangle$ we get the H-multiplied result $H|b\rangle$.

We shall now get into the aspects of the parallelization idea. There is a basic tactic without handling the matrix–vector multiplication which would be that of distributing only the Hamiltonian terms mentioned above, that is, the λ index in Eq. (10). Explicitly, one node will deal with $H_{\boldsymbol{B}}$, another node will address the $c^{+}_{\boldsymbol{B}}c_{\boldsymbol{a}}$ term, and so on. However, this plan is prone to poor scalability showing parallel slowdown already for 6 nodes with a speedup of only 1.5. This slowdown is perhaps due to load imbalance since not all of the Hamiltonian terms involve the same number of operations. The site–site interaction consists only of a few logical rules, but terms such as block–site or site–block have to iterate over tensor products. Even when we compare these last two terms there is also an imbalance because of roaming over fast and slow matrix indices.

A more efficient option consists of the distribution over the central ($\theta = 1, \ldots, \mathcal{M}$) loop of the matrix–vector multiplication on the diagonalization algorithm (Davidson in our case). Each task will

[2] This fixed-sized problem law neglects important effects such as overhead, cache effects, network latency, etc.

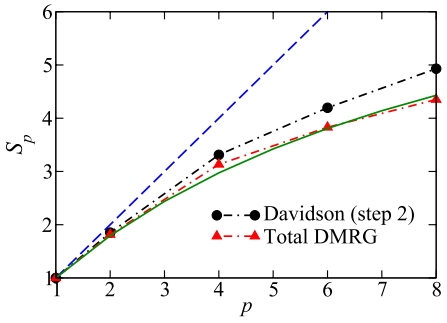

Fig. 2. Speedup scaling of a DMRG calculation for the ground state. Circles correspond to the Davidson algorithm (step 2 in Section 2) performance for the θ index distribution case ($\Sigma = 8.6(2)\%$). Triangles correspond to the total DMRG calculation of the ground state ($\Sigma = 11.5(3)\%$) with the corresponding Amdahl's law (full line). Ideal scaling is included for comparison (dashed line).

apply the full H to $\lfloor \mathcal{M}/p \rfloor$ states and the first $\mathrm{mod}(\mathcal{M}, p)$ tasks will handle an extra state.[3] We do not distribute the sub-blocks of the relevant operators labeled by their quantum numbers because of their dissimilar dimensions. With this strategy, we get values of speedup of 3.5 in an 8-node process with a serial fraction of 11.5%. To achieve an even faster realization when distributing over the θ index, one should also share out all of the linear algebra (`daxpy`, `ddot`, `dscal`, and `dcopy`) operations in the Davidson algorithm. These operations include orthonormalizations, inner products and the normalizations of the vectors added to the Davidson basis expanding the ground state $|\psi_0\rangle$. In doing so, we have now moved up the speedup to 4.9 on 8 nodes ($\Sigma = 8.6\%$). The scalability properties of the distributed version of the DMRG calculation for $|\psi_0\rangle$ are shown in Fig. 2. The load imbalance in this case goes as $p/\mathcal{M}$ which is negligible for actual DMRG simulations.

The performance properties of Davidson parallelization are strongly affected by the reduction operations of the matrix–vector multiplication, hence the better the implementation of these the better the speedup will be. This leading behavior could be diminished by ordering the superblock basis properly. This way, all of the reduction calls of order $\mathcal{M}$ are optimized by calls of order $\mathcal{M}/p$ or less. To show this, we have used a test block-diagonal matrix that does not require any reduction calls at all in the application of H. By doing this, we have obtained a serial fraction of $\Sigma = 0.87(2)\%$ (down to 20 processors) on the Davidson scheme, whereas when we consider the Hubbard Hamiltonian, we get a serial fraction of 8.6% as a result.

The most simple distribution one can think of was implemented in the rotation (decimation) of the operators relevant to H (such as $c_\uparrow$, $c_\downarrow$ for the Hubbard model case), that is, a row-distributed matrix–matrix multiplication. The final result is a serial fraction of 25% for this section of the algorithm. The reader should remember that Amdahl's law is a very simplistic proposal on the performance of a parallelized algorithm; serial fractions allow us to easily understand the results and what to expect of a distributed version of the serial code.

In order to better understand the performance obtained, we now make a comparison between our MPI implementation of the 1D Hubbard model and the shared-memory (OpenMP) version of the 2D Hubbard model [4]. The whole DMRG performance of the MPI implementation shows a better behavior than in the shared-memory version ($\Sigma = 11.5\%$ compared to $\Sigma = 16\%$ [4]) in spite of the fact that the Davidson algorithm results are not as good as previous ones ($\Sigma = 8.6\%$ compared to $\Sigma = 6.5\%$ [4]). This improved behavior could be related to the additional parallelization

[3] $\lfloor x/y \rfloor$ meaning the integer division and $\mathrm{mod}(x, y)$ stands for the modulo operation with x and y real numbers.

Table 1
Relative runtimes and serial fractions percentages at different steps of the algorithm. The unparallelized time (item *d*) is mainly consumed in building the *system* (or the *environment*), the density matrix and getting its spectrum.

Step	Time (%)	Σ (%)
a. Davidson algorithm	19.7	9.4(1)
b. Z_i and Z_iZ_j rotations	72.3	8.1(3)
c. H operators rotations	0.1	25(2)
d. Unparallelized sections	0.5	100(0)
e. Measurements	7.4	7.6(7)
f. Total calculation	100	9.4(1)

of the linear algebra operations mentioned above, added to the absence of collisions (and despite message passing) on the MPI algorithm or better communications originated on newer hardware improvements. Even though this comparison is not strictly valid because we are dealing with different geometries (1D versus 2D [4] Hubbard models), we must remark that our case is the worst case scenario. In 1D we have fewer Hamiltonian terms, meaning fewer independent processor operations in comparable Hilbert spaces with a similar amount of communications. This would suggest that for a more complex Hamiltonian (e.g. including longer range hoppings or different geometries such as 2D) our result for the serial fraction will be even smaller.

3.2. Novel strategy: correlation operators

If n-point correlations are required, the former distribution setup turns out to be insufficient because the ground state determination is not the longest time-consuming part anymore and is overtaken by the operator decimation and rotation (see Table 1). Therefore a new approach is mandatory to deal with that issue. The new strategy should take into account that the most time-expensive part is in this case the double matrix operation of the corresponding operators Z_i and Z_iZ_j (e.g. for $\mathcal{R}Z_i\mathcal{R}^+$: $Z_i\mathcal{R}^+$ and then $\mathcal{R}(Z_i\mathcal{R}^+)$). Typical correlation functions are the one-point and two-point functions [2], namely,

$$\langle Z_i \rangle = \langle \psi_0 | Z_i | \psi_0 \rangle,$$
$$\langle Z_i Z_j \rangle = \langle \psi_0 | Z_i Z_j | \psi_0 \rangle \tag{11}$$

with $i, j = 1, \ldots, L$ and L the length of the superblock chain. The number of (stored) matrices to be rotated (see Section 2, last step) at a given length calculation is $\mathcal{L} = \ell(\ell+3)/2$ (ℓ matrices coming from single-site operators Z_i and $\ell(\ell+1)/2$ coming from two-point correlation functions Z_iZ_j with $i < j$), with ℓ being the number of sites of the *system* or *environment* according to forward or backward sweeping. The correlations between the $\boldsymbol{B}$ and $\tilde{\boldsymbol{B}}$ blocks were calculated as a product of single-site operators in each block. The specific tasks involved in step 5 (see Section 2) are: (i) the reading of the current matrix Z_i from storage, (ii) the blocking step $Z_{[\boldsymbol{Ba}]} \longleftarrow Z_{[\boldsymbol{B}\oplus\boldsymbol{a}]}$, (iii) the two matrix–matrix products with the rotation matrix $\mathcal{R}$, and (iv) the corresponding saving of the new matrix $Z_i^{\mathrm{new}} = \mathcal{R}Z_i\mathcal{R}^+$.

We shall show below two ways to address this issue: a pool of tasks [18] and what we have called a *uniform-matrix distribution* (UMD) parallelization. In this latter strategy every node has almost the same load (see below) without a master node. The UMD parallelization seems to have a better output because it has fewer communications (only at the very beginning of the subroutine) and takes more advantage of the nodes available during the calculation (see below). The pool of tasks is a more elegant and common solution but in practice, a slower option. The speedup results for these two parallelized DMRG calculations of correlation functions are shown in Fig. 3. The efficiency for the UMD case is shown in Fig. 4.

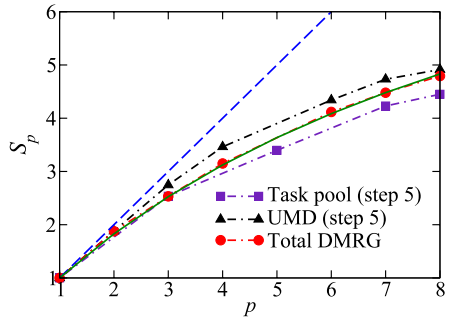

Fig. 3. Pool of tasks (squares) and UMD parallelization (triangles) performance for step 5 in Section 2. The values of the serial fractions were $\Sigma = 11.1(2)\%$ and $\Sigma = 8.1(3)\%$ respectively. The speedup factor corresponding to the total DMRG calculation using the UMD technique for the calculation of the correlation functions (circles) was $\Sigma = 9.4(1)\%$. Its corresponding Amdahl's law is also included (full line) and the ideal scaling is shown for comparison (dashed line).

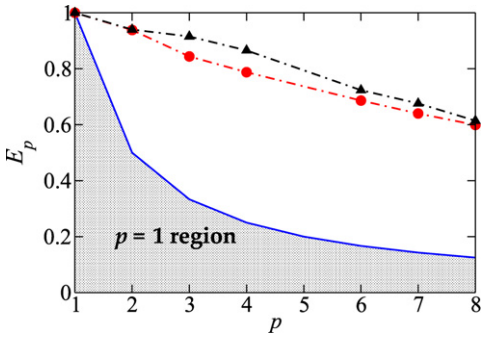

Fig. 4. Efficiency plot for the UMD case shown in Fig. 3 with the same symbol convention. The shaded region corresponds to the cases where no speedup is gained compared to the $p = 1$ case.

In the pool of tasks paradigm [18], the data to be processed (the Z matrices) are divided into small units with similar structures called *tasks*. All of these tasks form the so-called *task pool*. One node, the master process, manages this large amount of tasks, always sending to idle workers more work to do until all of the tasks have been executed (empty pool). This model is effective in situations where the available nodes have very different technical specifications, because the least loaded or more powerful hosts do more of the work and all of the hosts stay busy most of the runtime. The serial fraction obtained in this implementation was about 11.1% (see Fig. 3). The optimal result depends on the number of tasks in which the whole job is divided. If this number is too small, parallel slowdown will already appear. In addition, the greater the number of tasks the bigger the amount of communications will be.

Let us now explain the UMD technique. This distribution proves to be easier to code and more efficient than the pool of tasks. The key idea is to keep all of the processors on the same working settings so we can take full advantage of the accessible hardware. The distribution is performed in terms of blocks of contiguous local and non-local operators. If the number of processors is p then each processor stores $\lfloor \mathcal{L}/p \rfloor$ operators, except maybe the first mod $(\mathcal{L}, p)$ ones that will store $\lfloor \mathcal{L}/p \rfloor + 1$ matrices. Load imbalance in this case goes as $p/\mathcal{L}$, which is imperceptible for larger lattice lengths, i.e. larger $\mathcal{L}$. The serial fraction has now been improved to $\Sigma = 8.1\%$ (in the double quad-core system) as shown in Fig. 3.

There are many more communications in the pool of tasks compared to the UMD case. These communications are related to petitions coming from the workers involving statuses such as: "task done" and "ready to work"; and the complementary messages sent by the master node with the proper information about the task to be made. On the contrary, the UMD settings just need very few communications that keep track of the set of operators to be handled by each node. This message passing should be posted at the beginning of the corresponding iteration.

In both parallelization policies, if a given node demands a specific set of matrices that is not currently in local storage, an implemented queue manager handles this type of requests by sending the matching operator. This is done by means of a book-keeping of the matrices and its current owners throughout the entire cycle. Hence, when all of the desired matrices have been shipped, a new-owner message should be broadcasted to the rest of the active processors. The rotation matrix $\mathcal{R}$ is replicated along all of the nodes. This procedure allows each processor to save runtime by storing the new operators locally. For instance, if at some point through the simulation a processor, say, number 1 requests an operator that in an earlier step was assigned to processor, say, number 2, the queue handler transfers the required matrix from processor 2 to the corresponding node making an update of the owner matrix-bookkeeping. This procedure does not affect the task being performed by processor 2 avoiding synchronization delays. For the UMD case we have found a serial fraction of $\Sigma = 9.4\%$ in the star topology network.

The origin of the serial fraction of the presented parallelization schemes is perhaps due to the following factors: processes contending available cache space, racing conditions linked to the storage of the corresponding matrices, or the transfer of the requested data between processes. In order to reduce the total serial fraction of the whole process attention should be paid to the rotations of the operators (item *b* in Table 1), the Davidson algorithm (item *a*), and the unparallelized sections (item *d*). The measurements are discussed below. As for item *b*, the most time-expensive of all of the four steps at this point (addressed at the beginning of this subsection) would be consecutively: the two matrix–matrix products, the writing of the outcome to disk, the reading of the input from disk, the blocking operation and, in the star-topology case, the matrix copying among nodes. Unavoidable points are probably the I/O operations, the matrix multiplications, and the optimized blocking due to the use of symmetries. Therefore the candidate stage to be improved is the data transfer protocol (`ssh-server`) for the networking case. Using a socket-type communication or a remote server will certainly enhance the achieved speedup. As for the Davidson step, in all of the strategies, one could try to reduce the few synchronization calls with the consequence of having more local operations. And finally, the total serial fraction could be reduced further if some kind of parallelization scheme is implemented in the unparallelized section of item *d*.

There is a small discrepancy between the values of the serial fractions shown in Fig. 2 and Table 1, item *a* (with and without correlations) for the Davidson part. This may be due to the effect of the compilation when correlations are included. However, the values are compatible within the numerical error. Now, taking into account the Davidson diagonalization, as well as the Hamiltonian operators and the rotation of the operators to be measured, we should get a weighted average serial fraction of $\Sigma_{\text{total}} = 8.8\%$ as for the parallelized sections, but due to the unparallelized fraction of the code (item *d*) the final serial fraction is actually 9.4%. Finally, the corresponding distribution was done for the measurement part in the same way as for the distribution over the θ index in Eq. (10), with the exception that the $\mathcal{M}$-size vector `reduce` calls have been replaced by single-data reductions associated to the partial inner products $\langle\psi_0|Z|\psi_0\rangle$. The serial fraction for this section of the algorithm was $\Sigma = 7.6\%$. This value is probably related to the reading of the Z matrices from local or remote storage depending on the final Z-bookkeeping. It should be mentioned that this is just a minor optimization compared to the whole calculation, but it is rather straightforward to code this section of

the DMRG algorithm once that of the Davidson diagonalization has been implemented.

To estimate the performance of each node as compared to communication times, we show in Fig. 4 the parallel efficiency of the whole process in the UMD case. This quantity shows a very nice behavior up to the number of nodes used. For the $p = 8$ case E_p is around 60% meaning that each processor is actually working more than half of the total computational time. It also shows the good reliability of the parallelized algorithm suggested in this work. Parallel efficiency of a single-CPU is shown for comparison (continuous line). An improvement in the overall efficiency was observed when the number of states kept was increased ($m = 400$–1000), as expected from a non-fixed-sized parallel problem [19]. For instance, for $m = 1000$, E_p is increased by 20% for $p = 8$ with an overall serial fraction of $\Sigma = 5.5\%$. It should be noticed that the more operators are measured the more effective this novel strategy will be.

Simulations of ladder-type systems have shown that the ratio of runtimes between Davidson diagonalization and the rotation of the operators is not as remarkable as in the one-dimensional case. However, for long enough systems, the time of the rotation of the operators will be a significant part of the total time justifying the implementation of the present parallelization strategies. The change of the Davidson runtime stems from the increasing number of terms of H as pointed out in the previous subsection.

Lastly, in order to reproduce well-known results for the $S^z(q)$ and $N(q)$ structure factors [20], we have performed serial and distributed numerical simulations for a quarter-filled one-dimensional Hubbard chain of $L = 128$ sites with $m = 400$ states per block and an interaction parameter $U/t = 8$. Two sweeps for the finite-size algorithm and open boundary conditions were imposed in the calculation. The truncation error was $\epsilon_\rho \sim 10^{-7}$. The total runtime on a 1-node process was about 165 hours compared to, for instance, 33 hours on an 8-node process.

4. Conclusions

We have presented an efficient parallelized version of a DMRG code devoted to the calculation of n-point correlation functions. Unlike previous approaches, the current strategy was implemented in a passing message context (MPI) allowing for a better performance than for the shared-memory scheme. The overall serial fraction of the whole process was about 9.4% and the efficiency was around 60% up to eight nodes. In spite of the fact that our parallelization scheme does not scale well to hundreds of nodes it does allow simulations not reachable by serial coding with a maximum speedup of $1/\Sigma = 10.6$ according to Amdahl's law. Causes of parallel slowdown were addressed and possible ways of decreasing the serial fraction were presented.

Acknowledgements

J.R. would like to thank to E. Dari and E. Tapia for useful discussions and is infinitely indebted to P. Mateo for unconditional support. This work was done in the framework of projects PIP 5254 of the CONICET and PICT 2006/483 of the ANPCyT.

References

[1] S.R. White, Density matrix formulation for quantum renormalization groups, Phys. Rev. Lett. 69 (1992) 2863–2866;
S.R. White, Density-matrix algorithms for quantum renormalization groups, Phys. Rev. B 48 (1993) 10345–10356.
[2] K. Hallberg, New trends in density matrix renormalization, Adv. Phys. 55 (2006) 477–526;
U. Schollwöck, The density-matrix renormalization group, Rev. Mod. Phys. 77 (2005) 259–315.
[3] R.M. Noack, S.R. White, The density matrix renormalization group, in: I. Peschel, X. Wang, M. Kaulke, K. Hallberg (Eds.), Density-Matrix Renormalization: A New Numerical Method in Physics, in: Lecture Notes in Physics, vol. 528, Springer, Berlin, Heidelberg, New York, 1999.
[4] G. Hager, E. Jeckelmann, H. Fehske, G. Wellein, Parallelization strategies for density matrix renormalization group algorithms on shared-memory systems, J. Comp. Phys. 194 (2004) 795–808.
[5] B. Chapman, G. Jost, R. Pas, Using Openmp: Portable Shared Memory Parallel Programming, MIT Press, Cambridge, MA, 2007. Official website http://www.openmp.org/.
[6] G.K.-L. Chan, An algorithm for large scale density matrix renormalization group calculations, J. Chem. Phys. 120 (2004) 3172–3178.
[7] Y. Kurashige, T. Yanai, High-performance ab initio density matrix renormalization group method: Applicability to large-scale multireference problems for metal compounds, J. Chem. Phys. 130 (2009) 234114-1-21.
[8] S. Yamada, M. Okumura, M. Machida, Direct extension of density-matrix renormalization group to two-dimensional quantum lattice systems: Studies of parallel algorithm, accuracy, and performance, J. Phys. Soc. Jpn. 78 (2009) 094004-1-5.
[9] G. Alvarez, The density matrix renormalization group for strongly correlated electron systems: A generic implementation, Comp. Phys. Comm. 120 (2009) 1572–1578.
[10] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference – The MPI Core, Vol. 1, 2nd edition, MIT Press, Cambridge, MA, September 1998. MPI forum website: http://www.mpi-forum.org/.
[11] J. Hubbard, Electron correlations in narrow energy bands, Proc. Roy. Soc. A 276 (1963) 238–257;
J. Kanamori, Electron correlation and ferromagnetism of transition metals, Prog. Theor. Phys. 30 (1963) 275–289.
[12] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Stand. 45 (1950) 255–282.
[13] K. Wilson, The renormalization group: Critical phenomena and the Kondo problem, Rev. Mod. Phys. 47 (1975) 773–840.
[14] W.M.C. Foulkes, L. Mitas, R.J. Needs, G. Rajagopal, Quantum Monte Carlo simulations of solids, Rev. Mod. Phys. 73 (2001) 33–83.
[15] H. Fehske, R. Schneider, A. Weisse (Eds.), Computational Many-Particle Physics, Lect. Notes Phys., vol. 739, Springer, Berlin, Heidelberg, 2008, pp. 681–768.
[16] G.M. Amdahl, Validity of the single-processor approach to achieving large scale computing capabilities, in: Proceedings of AFIPS Spring Joint Computer Conference, 30, Atlantic City, NJ, 1967, pp. 483–485.
[17] E.R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, J. Comp. Phys. 17 (1975) 87–94;
C.W. Murray, S.C. Racine, E.R. Davidson, Improved algorithms for the lowest few eigenvalues and associated eigenvectors of large matrices, J. Comp. Phys. 103 (1992) 382–389.
[18] M. Korch, T. Rauber, A comparison of task pools for dynamic load balancing of irregular algorithms, Concurrency Computat.: Pract. Exper. 16 (2004) 1–47.
[19] J.L. Gustafson, Reevaluating Amdahl's law, Commun. ACM 31 (1988) 532–533.
[20] R.M. Noack, S. Daul, S. Kneer, Properties of the Hubbard chain, in: I. Peschel, X. Wang, M. Kaulke, K. Hallberg (Eds.), Density-Matrix Renormalization: A New Numerical Method in Physics, in: Lecture Notes in Physics, vol. 528, Springer, Berlin, Heidelberg, New York, 1999.