Improving CASINO performance for models with large number of electrons

Lucian Anton[1,*], Dario Alfe[2,†], Randolph Q. Hood[3,‡], David Tanqueray[4,§]

[1] *Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, UK*

[2] *Department of Earth Sciences, and Department of Physics and Astronomy, University College London, Gower Street, London,WC1E 6BT, United Kingdom*

[3] *Lawrence Livermore National Laboratory, 7000 East Ave. L-045 Livermore, CA 94550, USA*

[4] *Cray UK Limited, 2 Brewery Court, High Street,Theale, Reading, Berks, RG7 5AH, UK*

May 1, 2009

Abstract

Quantum Monte Carlo calculations have at their core algorithms based on statistical ensembles of multidimensional random walkers which are straightforward to use on parallel computers. Nevertheless some computations have reached the limit of the memory resources for models with more than 1000 electrons because of the need to store a large amount of electronic orbitals related data. Besides that, for systems with large number of electrons, it is interesting to study if the evolution of one configuration of random walkers can be done faster in parallel. We present a comparative study of two ways to solve these problems: i)distributed orbital data done with MPI or Unix inter-process communication tools, ii)second level parallelism for configuration computation.

Keywords: Quantum Monte Carlo, shared data, Unix interprocess communication, parallel algorithms.

1 Introduction

1.1 Background

Quantum Monte Carlo(QMC) methods are one of the most precise tools used in computing physical properties of system that are made of a relatively large number of atoms: crystals, nanoclusters or macromolecules. Although QMC has the advantage of scaling with second or third power of the system size, very precise results need a large amount of sampling of the phase space and therefore the most challenging QMC problems need to use the most performant hardware and available algorithms [1].

In this paper we present a series of algorithmic improvements implemented in the past year

*email: lucian.anton@nag.co.uk
†email: d.alfe@ucl.ac.uk
‡email: hood9@llnl.gov
§email: dt@cray.com

on CASINO, which is a QMC software package developed and maintained over the last 10 years at Cavendish Laboratory, Cambridge University, UK [2, 3].

In order to fix the terminology we describe briefly the basic mathematical concepts that provide the foundation of QMC algorithms, for a more detailed presentation we direct the reader to Refs [3, 4].

A typical quantum many-body system has N_e electrons with positions $\mathbf{R} = \{\vec{r}_e\}$, of which $N_\uparrow$ have spins up and $N_\downarrow = N_e - N_\uparrow$ have spins down, and N_I ions with positions $\mathbf{R}_I = \{\vec{r}_I\}$. The particle interaction is described by the quantum Hamiltonian

$$H = -\sum_{i=1,N_e} \frac{\hbar^2}{2m}\nabla^2_{r_i} + V(\mathbf{R}, \mathbf{R}_I) \ ,$$

from whose eigenstates and eigenvalues one can compute in principle all physical quantities describing the system. For realistic Hamiltonian operators exact solution are not available but good physical results can be obtained with approximate one particle solution, the most successful technique in this class being the Density Functional Theory.

CASINO calculations improve further the one particle solutions adding to them particle correlation contributions with the help of the following two QMC methods:

1.1.1 Variational Monte Carlo (VMC)

VCM calculations use a trial function $\Psi(\alpha, \mathbf{R}, \mathbf{R}_I)$ whose parameters α are determined from the minimization of the average energy energy or its variance:

$$E(\alpha) = \frac{\langle\Psi(\alpha)|H|\Psi(\alpha)\rangle}{\langle\Psi(\alpha)|\Psi(\alpha)\rangle} \ . \qquad (1)$$

A typical trial function for an electronic system is a product between the Slater determinants of one particle orbitals multiplied by a function that encapsulates the particle correlations.

$$\Psi(\alpha, \mathbf{R}, \mathbf{R}_I) = e^{J(\alpha,\mathbf{R},\mathbf{R}_I)} D_\uparrow(\mathbf{r}_1, \ldots, \mathbf{r}_{N_\uparrow}) \qquad (2)$$

$$\times D_\downarrow(\mathbf{r}_{N_\downarrow+1\ldots,\mathbf{r}_{N_e}}) \ , \qquad (3)$$

where $D_{\uparrow,\downarrow}$ are the Slater determinants of the electrons with spin up($\uparrow$) or down($\downarrow$). The simplest Jastrow factor $J(\alpha, \mathbf{R}, \mathbf{R}_I)$ is a sum of two-electron functions

$$J(\alpha, \mathbf{R}, \mathbf{R}_I) = -\sum_{\substack{i>j \\ \sigma_i,\sigma_j}} u_{\sigma_i,\sigma_j}(\alpha, |r_i - r_j|) \qquad (4)$$

where u is taken from the homogeneous electron gas. Over the years more elaborated extensions of $J(\alpha, \mathbf{R}, \mathbf{R}_I)$ have been developed in order to include three electron correlations and electron ion correlations [4].

Many studies it have shown that VMC calculations recover up to $70-90\%$ of the correlation energy but the remaining contributions are hard to conquer because the results cannot be improved systematically in the VMC framework. As a matter of fact the main use of VMC calculations in CASINO is to generate the configuration needed for the Diffusion Monte Carlo calculation.

1.1.2 Diffusion Monte Carlo (DMC)

DMC calculation is based on the observation that Schrodinger equation in imaginary time ($t = i\tau$),

$$-\frac{\partial\Psi(\mathbf{R},\tau)}{\partial\tau} = -\frac{1}{2}\nabla^2\Psi(\mathbf{R},\tau) + (V(\mathbf{R}) - E_T)\Psi(\mathbf{R},\tau) \ . \qquad (5)$$

is a diffusion equation plus a source/sink term given by the potential. If the parameter E_T is tuned to the groundstate energy the trial wavefunction function is projected to the ground state of the Hamiltonian. The anti-symmetric nature of the electronic wavefunction poses serious problems for the numerical solution of Eq (5). A way out of to this difficulty is to introduce the so called fixed-node approximation, in which a modification of Eq (5) is used that projects the wavefunction onto the ground state wavefunction which has the same nodal surface of a trial wavefunction. In practice is was found that this is a very good approximation.

1.2 A Survey of CASINO algorithm and the memory problem

At its core CASINO algorithm uses configurations of N_e three dimensional random walkers (RW) which evolve according to the distribution probability associated to wave trial function for VMC calculation, Eq (1), or to the Schrodinger equation for DMC calculation , Eq (5).

A typical CASINO calculation involves the following main steps:

- read the input parameters and the needed data such as orbital data coefficients, Jastrow parameters,
- run a VMC calculation to generate the RW configurations needed for DMC,

- run a DMC calculation: evolve the configuration, compute the local energy and decide on branching, accumulate statistics of the observable quantities.

The computation proceeds by moving one walker at a time, the transition probability at each step depends on the current values of the Jastrow factor and one particle orbitals (OPO) at the current position of all random walkers. The orbitals can be represented using various basis sets, including plane waves, Gaussians, or B-splines. In this article we are concerned with the representation in B-Splines [5], which are localised third order polynomials sitting on a three-dimensional grid in real space. They share the same properties of plane-waves as being systematically improvable and unbiased, but they are localised, and as such a factor $1/N_e$ more efficientthan plane waves: for each point in space there are always only 64 B-Splines that have non-zero values. The evaluation of each orbital therefore only requires the evaluation of the value of 64 B-splines, and fetching from memory of the corresponding 64 coefficients.

In the program the B-Splines coefficients (BC) are stored in a rank five array $a(1 : N_b, 0 : N_{gx}-1, 0 : N_{gy}-1, 0 : N_{gz}-1, N_s)$, where N_b, N_{gx}, N_{gy}, N_{gz}, N_s are the number of orbitals, the number of the of grid points in the three spatial direction and the number of spins, respectively.

The amount of BC needed in computation is determined by two factors: i) For each spin value the the number of orbitals must be equal with the number of electrons with that spin, ii) the grid spacing is determined by the precision of the one particle calculation, the higher the precision the finner the grid must be.

This two requirements conspire to create a large amount of BC. For example, if we consider a system with 1000 electrons, split in half spin up, half spin down, we need at least 500 one-particle orbitals for a non-magnetic system since in this case one can use the same set of BC for both spins. The spatial grid can reach or exceed 80 points in each direction, hence, for the previous quoted numbers one needs approximately 2 GB of memory, if the values of BC are stored in double precision. This amount of memory is close to the maximum available memory of the currently used processors.

In the following sections we present a study of several solutions we have tried to this problem: In section 2 we describe the algorithms used for shared BC and the results of the performance measurements for each algorithm. In section 3 the second level parallelism algorithm is described and its performance is analysed. In section 4 we discusses some other improvements applied to the code and the general conclusions.

We mention that the performance measurements runs were done on HECToR, which is a Cray XT4 based in UK and used by the academic and scientific research communities [7]. We have tested the code on the dual core AMD Opteron which equips the system in phase I and on the quadcore AMD Opteron that will be used on phase II of HECToR. The dual core processor has the following technical specifications: 2.8 GHz clock rate, 6 GB of RAM, 64KB L1 cache, 1MB L2 cache, peak performance 11.2 Gflops in double precision. The quadcore processor has the following technical specifications: 2.3 GHz clock rate, 8 GB of RAM, 64 L1 cache, 512 KB L2 cache, 2 MB L3 cache(shared), peak performance close to 40 Gflops in double precision. The code was compiled with PGI v8.0.2.

2 Sharing large BC data sets

In the initial algorithm of CASINO each task has a copy of the BC needed to compute the orbitals values. Since the BC sets are identical on each task and their values do not change during computation the obvious solution to the memory problem is to share the data among groups of tasks, especially when the hardware provides shared memory.

In this section we present the main features of three solutions for BC sharing which are schematically illustrated in Fig 1 and discussed in the following.

2.1 Shared memory on a processor or node (SHM)

At the time of writing MPI standard does not offer the possibility to address a common memory segment for a group of tasks on shared memory systems. Nevertheless shared memory can be used with the help of the Unix System V inter-process communication functions shmget, shmat [6], which allocate a memory segment and attach it to the memory space of each calling task.

The main steps this algorithm are as follow:

1. find the task belonging to the same processor or node (multiple processors that share

memory) with "mpi_get_processor_name" procedure,

2. allocate the memory needed to store the BC and attach it to the tasks memory space with shmget and shmat,

3. pass the reference of the shared memory segment to the FORTRAN program via a Cray pointer.

The implementation uses a set of functions written in C which group the tasks that can use shared memory and create the shared memory segment for them. The subroutines that make available the shared memory to a FORTRAN pointer using Cray pointers are provided in a FORTRAN module.

There are two non-portable components in this implementation according to CASINO coding standard (FORTRAN95+MPI), although widely available on parallel computer systems: the shared memory subroutines, which are not portable on a non-Unix operating systems, and the Cray pointers, which are not FORTRAN 95 compliant.

In order to conform with CASINO portability requirements or for use in cases in which shared memory is not needed an alternative FORTRAN module is provided that implements the initial algorithm in which each task has a full copy of the BC. The user can switch between the two modules with the help of a makefile variable(CASINO does not use code preprocessing).

2.2 MPI two-sided data transfers (MPI-2S)

This algorithm is fully compliant with FORTRAN 95 as it uses two-sided MPI calls for BC transfers between tasks.

The main steps of the algorithm works as follow:

1. The total number of tasks of a given computation is split in groups of size n_g.

2. The BC are distributed among each group of tasks in the following manner: each task has the full spatial grid and N_b/n_g orbitals picked with a periodic rule.

3. When a task needs to compute the orbitals' values for a given electron it broadcasts the electron's coordinates to its group members and waits for them to return the orbitals' values whose BC are stored on each of them.

Because each task evolves its configuration randomly a synchronization mechanism must be provided. In the current implementation it consists in additional 'sentinel calls' in the inner loop of the configuration computation that answer the requests of orbital computation from the associated tasks.

2.3 One-sided data transfers (MPI-1S,SHMEM)

This is a variation study of MPI-2S that tries to avoid the synchronisation delays of the previous algorithm using one-sided data transfer provided by MPI or SHMEM libraries. In this case the task that reaches the orbitals computation sector transfers the set of BC from the memory of the associated tasks without the need of a matching call on their side. This algorithm has two drawbacks: i)the amount of data to transfer between two tasks is 64 times larger than in the case of the orbital transfer, ii) the data set to be transferred has non-contiguous memory addresses because it is a 4×4 block of the spatial grid.

2.4 Tests results

In Table 1 we present the execution times of the orbital computing subroutine for the discussed algorithms. We collected data from a short run (12 time steps) but which yields more that 10^3 timing points for the observed sections. The data were collected in the DMC section using the internal timer of the code. We present also the total time taken by the DMC calculation. The electronic system has 1024 electrons and the BC size is approximately 2.4 GB.

As expected, the results show that SHM algorithm is by far the most efficient since it avoids unnecessary data transfers between tasks. MPI two-sided looks as an acceptable alternative when shared memory is not available. The weak performance of the MPI one-sided algorithm deserves some further comments. Although at first sight the one-sided MPI features would seem to be asynchronous, there is no requirement in the MPI specifications that they should be implemented as such, and indeed in the MPICH implementation they are all saved up and performed together at the next collective or sync call, where they are handled internally as part of the underlying 2-sided MPI communications design. The SHMEM calls on the other hand are usually performed asynchronously being built directly on top of the XT Portals library which does provide some

degree of asynchronous support, and in fact the results show that SHMEM algorithm performance is slightly better than the MPI-2S algorithm.

3 Second level of parallelism

The second level parallelism(SLP) algorithm seeks to increase the computation speed by employing more that one task to move one RW configuration to its next state. As the computation time of a configuration in CASINO scales with N_e^p, with $p = 2, 3$, the computation time per configuration for a system with $\mathcal{O}(10^4)$ electrons could be more 100 times longer than a for a system with $\mathcal{O}(10^3)$ electrons, which is the maximum used value at the present. SLP solves also the large size BC problem as it distributes the BC data among the group of task that perform the computation for one configuration.

As in the case of MPI-2S algorithm SLP divides the tasks in groups, named pools, of given size (typically 2 or 4). At start the program reads the BC and distribute them among the pool members similar to MPI-2S algorithm. The difference is that only one configuration is computed at a time by all the tasks belonging to a pool. One of the tasks, named "pool head", controls the computation and sends signals to the other tasks about the next step of the computation. In this manner the synchronisation problem is removed and the pool's tasks can be used to compute more quantities beside the orbitals: sums that appear in the Jastrow factor, the potential energy and linear algebra operations needed for the Slater matrices.

We analyze the efficiency of SLP algorithm in the following way: for a pool of size n in the ideal case the computation time of one configuration would be $t_n = t_1/n$. However the communication time between tasks is not negligible and the work is not equally distributed over the pool's tasks because there are computations done only on the pool's head. We can measure the efficiency of a pool usage with the following parameter:

$$\eta = \frac{t_1/t_n - 1}{n - 1} \tag{6}$$

where η takes value 1 in the ideal case, 0 if the use of SLP does not increase the computation speed and becomes negative if the computation time increases with the pool size.

In Table 2 we present the computation times for three sections that are done in parallel: one particle orbitals(OPO), Jastrow function and the Ewald sum as well as for the whole (DMC) section; the pool sizes are $1, 2, 4$. The input file is identical to that used for shared memory measurements, see Table 1. The efficiency parameter shows that the best ratio computation/communication is obtained in the subroutine that computes the orbital values on dual core processor. The quadcore processor is faster and consequently the weights of the communication time increases. There is a significant degradation in efficiency between pools of size 2 and 4 for orbitals and Jastrow factor, but for Ewald summation increases slightly. It is also worth mentioning also that efficiency of a quad core processor is overall not significantly larger than that of two dual core processors. We note also that the efficiency of the calculation OPO increases for the larger system.

The overall efficiency in DMC sector of the current implementation is rather small as the computations of the Slater determinant and of the associated matrices are done on the pool's head. The Slater determinants are computed in two ways in CASINO: i) using LU factorization of the Slater matrix, which scales as N_e^3, ii) using an iterative relationship for the cofactor matrix [8], which scale as N_e^2 but is numerically unstable. We have implemented a parallel computation over the pool cores of this two subroutines for VMC calculations using Scalapack subroutines. The timing results, Table 3, show an excellent scaling for the N_e^3 algorithm but little improvement or slight degradation for N_e^2 algorithm which in fact is the most important as the ratio of calls between the two subroutines is small multiple of N_e in the favor of N^2 algorithm.

4 Final remarks and conclusions

In other developments we mention that one of the authors (RH) has increase significantly the orbital computation speed after reordering the BC array storage to a pattern which increase data locality and uses efficiently the cache memory. Also he has shown that using single precision BC alleviates the memory requirements and improves the computation speed.

The performance of BC data input at the beginning of the computation has been improved significantly. Initially the BC were stored in an ASCII file and for large sets the reading time could reach 30 minutes. In the current version of CASINO the

BC data can be converted to binary files using FORTRAN binary or MPI input/output library and the new input time is of the order of tens of seconds.

In conclusion, CASINO performance has been significantly increased by implementing better algorithms that can take advantage of the useful features of the hardware and libraries used at the moment on the high performance computing systems.

Two solution for the large size BC have been implemented in CASINO: SHM and MPI-2S. The SHM version is the fastest but it works only if shared memory is available via Unix interprocess communication tools. The MPI two-sided, although slower, is fully FORTRAN 95 compliant and the sharing of the BC is not constrained by the presence of shared memory.

The SLP study provides a solution for faster computation of very large configurations of electrons. Efficiencies up to 50% has been achieved in parallel sector of the computation, but an algorithm with better scaling is needed for the iterative computation of the Slater determinant.

Acknowledgements

One of the authors (RH) acknowledges that his work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] Kenneth P. Esler *et al*, Journal of Physics: Conference Series **125** 012057 (2008).

[2] `http://www.tcm.phy.cam.ac.uk/ mdt26/casino2.html`

[3] Richard Needs, Mike Towler, Pablo Lopéz Ríos, CASINO User's Guide (Theory of Condensed Matter Group, Cavendish Laboratory, Cambridge, UK, 2008).

[4] M. D. Towler, Phys. Stat. Sol. (B) **243**, No. 11, 2573 (2006).

[5] D. Alfè, M. J. Gillan, Phys. Rev. B **70** 161101(R) (2004).

[6] Unix man pages.

[7] `http://www.hector.ac.uk`

[8] S. Fahy, X. W. Wang and Steven G. Louie, Phys. Rev. B **42** 3503 (1990).

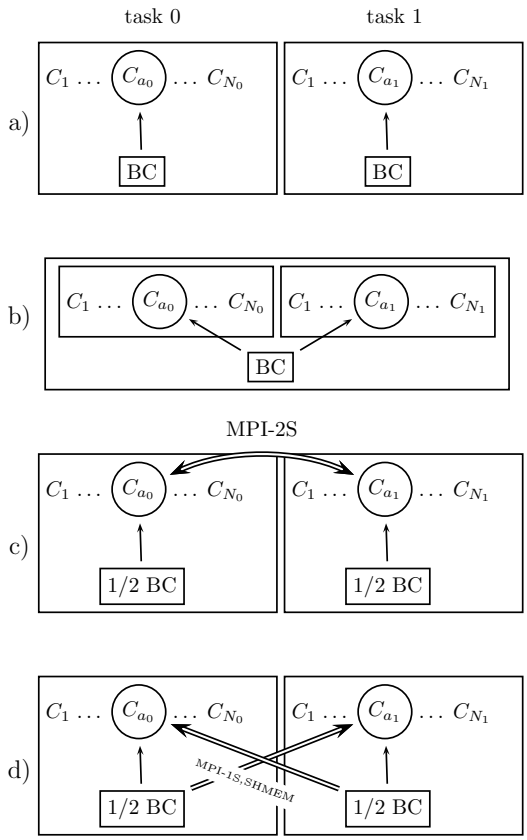

Figure 1: Illustration of the BC data access patterns for two tasks in various algorithms. Each task computes in serial mode a list of configurations $C_1 \ldots C_{N_t}$ using the stored BC: a) The standard version, b) SHM version (shared memory needed), c) MPI two-sided version: the BC data are transferred using MPI two-sided communication, d) one-sided data transfers that can be implemented with MPI or SHMEM.

CPU	2DC		4DC		4QC	
	OPO	DMC	OPO	DMC	OPO	DMC
SHM	130	921	–	–	139	882
MPI-2S	371	1184	806	1458	546	1249
MPI-1S	562	1380	1669	2565	1430	2210
SHMEM	210	975	771	1759	536	1271

Table 1: Execution times in seconds for BC sharing algorithms described in Sec 2. The columns are organized as follow: 2DC shows data of runs made on 2 tasks using one dual core processors, 4DC is for runs of 4 tasks on 2 dual core processor and 4QC for 4 tasks using one quadcore processors. The OPO column shows the time spent for one particle orbital computation, DMC is the time for the whole diffusion Monte Carlo computation.

	DC			QC		
pool size	1	2	4	1	2	4
System 1	1024 electrons					
OPO	118	80(0.46)	78(0.17)	141	93(0.52)	64(0.40)
Jastrow	271	218(0.24)	189(0.14)	199	151(0.32)	123(0.21)
Ewald	79	59(0.34)	34(0.44)	122	90(0.36)	52(0.45)
DMC	773	656(0.18)	592(0.10)	743	610(0.22)	518(0.14)
System 2	1536 electrons					
OPO	267	171(0.56)	143(0.29)	311	197(0.58)	136(0.43)
Jastrow	640	505(0.27)	473(0.12)	454	340(0.36)	317(0.14)
Ewald	183	137(0.34)	81(0.42)	276	207(0.33)	121(0.43)
DMC	1965	1709(0.15)	1531(0.09)	1847	1522(0.21)	1358(0.12)

Table 2: Computing times in seconds for SLP algorithm for dual core (DC) and quadcore (QC) processors with pools of sizes 1, 2 and 4. The times are for the three section of the code that are computed in parallel (one particle orbitals, Jastrow, Ewald) and for whole DMC segment that contains also sections executed in serial mode on the pool's head. In brackets are the values of the efficiency parameter defined by with Eq (6).

	DC		
pool size	1	2	4
System 1	1024 electrons		
det N^3	12	5.5(1.18)	4.7(0.52)
det N^2	94	106(-0.11)	76(0.08)
System 2	1536 electrons		
det N^3	39	18(1.17)	10(0.90)
det N^2	330	344(-0.01)	231(0.14)

Table 3: Computing times in seconds for Slater determinants using Scalapack over the pool. The order N_e^2 method shows longer times since it is called a small multiple of N_e more times than the other. In brackets are the values of the efficiency parameter defined by with Eq (6).