Improving the parallelisation and adding functionality to the quantum Monte Carlo code CASINO

Lucian Anton
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, UK,
email: lucian.anton@nag.co.uk

September 28, 2009

Abstract

This report presents the results of a one year distributed Computer Science and Engineering (dCSE) support project for the code CASINO which has been focused on three main issues: i) better memory utilisation using shared data between tasks belonging to the same processor, ii) acceleration of the computation speed by implementing second level parallelism and iii) improved input reading speed for large sets of initial data.

Contents

1 Introduction

1.1 Background

Quantum Monte Carlo(QMC) methods are accurate numerical tools used for computing the properties of physical models that contain a relatively large number of atoms, e.g.: crystals, nanoclusters or macromolecules. Although QMC computing time has the advantage of scaling with second or third powers of the system size, very precise results require the need to process large samples of phase space configurations and therefore the most challenging QMC problems require use of the most performant hardware and available algorithms [1].

In order to set the terminology we shall briefly describe 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 $N_\downarrow = N_e - N_\uparrow$ 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 Hamiltonians exact solutions are not available but good physical results can be obtained with approximate one particle solutions, the most successful technique in this class being that of Density Functional Theory (DFT).

QMC calculations can further improve a one particle solution by providing particle correlation contributions with the help of the following two QMC methods:

1.2 Variational Monte Carlo (VMC)

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

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

A typical trial function for an electronic system is the product between the Slater determinants of one particle orbitals (OPO), obtained from a DFT calculation, multiplied by a function that describes the particle correlations.

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

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

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

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

where u is taken from the homogeneous electron gas. Over the years more elaborate 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 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 set needed for the Diffusion Monte Carlo calculation.

1.3 Diffusion Monte Carlo (DMC)

The DMC method is based on the observation that the 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 branching/annihilation term given by the potential. If the parameter E_T is tuned to the groundstate energy the trial wavefunction 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. Practical computations show that this is a very good approximation.

1.4 A survey of CASINO and of the dCSE problems

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

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

A typical calculation involves the following main steps:

- read the input parameters and the input data such as orbital data coefficients, Jastrow parameters,
- run a VMC calculation to generate the RW configurations needed for DMC,
- run a DMC calculation: propose a change of a configuration, compute the local energy and decide on branching, accumulate statistics of the physical quantities.
- if the computation is done in parallel a redistribution of the configurations amongst tasks is needed in order to keep the work evenly distributed.

Two main factors determine CASINO's performance for the function of the number of electrons in a given model: i)the memory needed

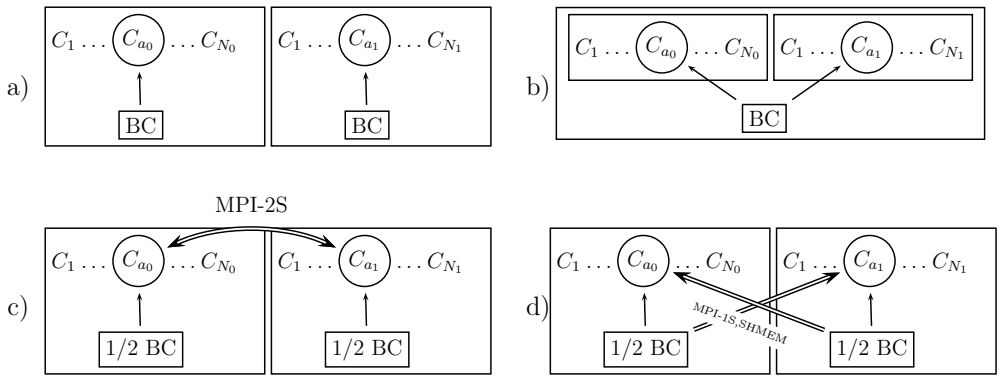

Figure 1: Illustration of the BC data access patterns for two tasks in various algorithms. Each task computes in serial mode a list of configurations $C_1 \ldots C_{N_t}$ using the stored BC: a) The standard version, each task has it own BC data set, b) BC are located on a sector of shared memory, c) 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.

to store orbitals values for the Monte Carlo algorithm and ii) the scaling of the computation time with the electron number [2].

Besides these problems it was found in practice that for parallel computation of models containing more than 1000 electrons and for computations with or on more that 5000 tasks there is an extremely long reading time of the orbitals' data or of the previously stored configurations (30 to 60 minutes).

In the following sections of this dCSE report we shall present in detail the proposed solutions for the above problems and discuss the performance gains they bring to the code. Section 2 describes the algorithms used for shared OPO data and the results of the performance measurements for each algorithm. Sections 3, 4 presents the second level parallelism algorithms and their performance is analysed. Section 5 presents the improvements for the reading of initial data. The benchmark tests were done on the dual core AMD Opteron which were used in phase I of HECToR and on the quad core AMD Opteron that are currently in use in 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 quad core 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.{2,6}.

2 Sharing large data sets

In this section we present the main features of three solutions proposed for orbitals' data sharing which are schematically illustrated in Fig 1.

A CASINO 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 OPO at the current position of all random walkers. The orbitals can be represented using various basis sets, including plane waves, Gaussian, or B-splines. In this report 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 which spans the whole physical system. 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 efficient than plane waves: for each point in space there are always only 64 B-Splines that have non-zero values. Therefore the evaluation of each orbital requires only the computation of 64 B-splines, which is much less than the total number of plane wave functions for a system with a large number of electrons (the number of plane wave functions scales with the N_e).

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 three spatial directions and the number of spins, respectively.

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

The above 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, which is close to the maximum available memory per core for the processors used on HECToR.

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.

2.1 Shared memory on a processor or node (SHM)

At the time of writing the 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 [7], which allocate a memory segment and attach it to the memory space of each calling task.

The main steps to this algorithm are as follows:

1. find the tasks 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 [6].

There are two non-portable components in this implementation according to the CASINO coding standard (FORTRAN95+MPI), although widely available on currently used parallel computers: the shared memory subroutines, which are not portable on a all 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 that do not need shared memory an alternative FORTRAN module is provided that implements the initial algorithm, that is, 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 was implemented in CASINO by Randolf Hood of LLNL in 2008. We have presented its main features in this report

because it was used in the performance tests. The algorithm uses two-sided MPI calls for BC transfers between tasks, its main steps are 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 configurations randomly a synchronisation mechanism must be provided. In the current implementation it consists of additional 'sentinel calls' in the inner loop of the configuration computation that answer the requests of orbital computations 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 can access the set of BC it needs 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 $4 \times 4 \times 4$ block of the spatial grid.

2.4 Test results

Table 1 presents the execution times of the orbital computing subroutine for the discussed algorithms. The data were collected from a short run (12 time steps), but which yields more than 10^3 timing points, inside the DMC section using the internal timer of the code. The total time taken by the DMC computation is also presented. The electronic system has 1024 electrons and the BC size is approximately 2.4 GB.

As expected, the results show that the SHM algorithm is by far the most efficient since it avoids unnecessary data transfers between tasks.

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 organised as follow: 2DC shows data for runs done on 2 tasks using one dual core processors, 4DC is for runs of 4 tasks on 2 dual core processors and 4QC for 4 tasks using one quadcore processor. The OPO column shows the time spent for one particle orbital computation, DMC is the time for the whole diffusion Monte Carlo computation.

MPI-2S appears to be an acceptable alternative when the amount of data surpasses the available shared memory (this may happen in the case of some disordered models). The weak performance of the MPI one-sided algorithm deserves some further comments provided by David Tanqueray: "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 with MPI

The second level parallelism (SLP) algorithm seeks to increase the computation speed by employing more than 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 size reached by the current calculations. As byproducts SLP solves the large size BC problem because it distributes the BC data among the group of task that perform the computation for one configuration and improves the load balance of the parallel

computation because the relative difference between the number of configurations on different pools decreases with the pool size (for a calculation with a fixed number of configurations per task or pool of tasks).

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 of MPI-2S algorithm is removed and the pool's tasks can be used to compute in parallel 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 analyse the efficiency of SLP algorithm in the following way: in the ideal case for a pool of size n 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 over the pool: one particle orbitals (OPO), Jastrow function, Ewald sum and also 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 efficiency is obtained for pools of size 2 for OPO computation. In the case of pool of size 4 the OPO computation is clearly more efficient on quadcore processor but the other quantities have similar performance, though slightly better on quadcore. 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

	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 (OPO, 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).

LU factorisation of the Slater matrix, which scales as N_e^3, ii) using an iterative relationship for the cofactor matrix [9], 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, shows an excellent scaling for the N_e^3 algorithm but little improvement or slight degradation for N_e^2 algorithm which in fact has a much larger weight in the computation time (in the last version of CASINO LU computation of the Slater determinant is call by default every 100,000 time steps).

4 Second parallelism level with OpenMP

The previous study shows that the parallel computation of a configuration across processors is rather expensive while the performance of distributed computation on the same processor is promising. In phase II HECToR is equipped with quadcore processors and it is expected that the next stages of the HECToR service will use processors with higher number of cores or/and shared memory for the processors belonging to a blade. In this hardware framework the mixed mode programming is the straightforward option for the implementation of SLP. OpenMP can be used to accelerate the computation of one con-

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

Table 3: Computing times in seconds for Slater determinants using Scalapack over the pool for the recursive algorithm, $O(N^2)$, or by LU factorisation, $O(N^3)$. The $O(N_e^2)$ method shows longer times because its number of calls is a small multiple of N_e larger that of the $O(N^3)$ method. In brackets are the values of the efficiency parameter defined by with Eq (6).

figuration, while keeping separated MPI tasks for separated sets of configurations.

OpenMP parallelism at loop level in CASINO is relatively easy to implement as the code contains patterns of nested loops over the number of electrons for the computation of various physical quantities which are sums of function that depend on two or more electrons coordinates.

The OpenMP parallelism is implemented for the inner loops of the computation. The external loop parallelism, which from a theoretical point of view should be more efficient, is in practice very hard to implement with OpenMP for two main reasons:

1. the code uses a buffering algorithm for various quantities to avoid the repetitive computation of the same quantity but which introduces data dependency,
2. most importantly at the moment, the current implementation of the OpenMP specifications in the used compilers are struggling to handle procedure calls that use module variables. PGI and Pathscale compilers have difficulties with module variables which also can be thread private (this problem disappeared in PGI at version 8.0.6). GCC compiler (version 4.3.3) crashes during compilation if the parallel region is contained in an internal subroutine (problem solved in version 4.4.0). Sometimes it is possible to rewrite the code in order to use the implemented OpenMP specifications, but this kind of solutions are time consuming and interferes unnecessarily with the work of the other developers. Ideally OpenMP at the loop level should not change the serial code.

threads	No MP	1	2	4
System 1		1024 electrons		
OPO	102	105	73(0.38)	61(0.56)
Jastrow	246	302	216(0.14)	170(0.15)
Ewald	79	78	39(1.03)	20(0.98)
$\bar{D}$	49	47	19(1.58)	9(1.48)
R_{ee}	123	114	65(0.89)	40(0.69)
DMC	672	723	481(0.40)	363(0.28)
System 2		1536 electrons		
OPO	218	232	166(0.31)	147(0.16)
Jastrow	539	648	470(0.15)	420(0.09)
Ewald	176	176	90(0.96)	46(0.94)
$\bar{D}$	178	180	126(0.41)	124(0.15)
R_{ee}	265	249	143(0.85)	84(0.72)
DMC	1542	1654	1154(0.34)	966(0.20)

Table 4: Computing times in seconds for OpenMP SLP algorithm for quad-core processors for 1, 2 and 4 threads. For comparison the first column shows timing data of the executable compiled without OpenMP flags. The times are for the five section of the code that are computed in parallel (OPO, Jastrow, Ewald, update of the $\bar{D}$ matrix, the electron-electron distances R_{ee}) and for the full DMC sector. In brackets are the values of the efficiency parameter defined by Eq (6) where t_1 is the taken from the first column.

The benchmarks test for the OpenMP parallelism was done on the same system as in the section 3. However a direct comparison between the numerical values is not straightforward as the CASINO code has undergone significant changes between the two measurements, nevertheless one can use the efficiency Eq (6) parameter for a performance comparison between the two SLP algorithms.

At a first sight two features need to be discussed regarding the values presented in table 4.

The efficiency of the OpenMP parallel loops varies. There are sections as Ewald and $\bar{D}$ for "System 1" which show larger than one efficiency and a very small overhead. On the other hand the overhead for the parallel loops in Jastrow sector is very large in both sectors and the scaling is rather poor. The extreme case of poor scaling is shown by the computation with four threads of the $\bar{D}$ for "System 2" which shows practically no speed gain, despite the fact that "System 1" has a perfect behaviour in this case. The main suspect of this variation is cache memory utilisation as the linear size of $\bar{D}$ matrix of "System 2"

is 50% larger than that of "System 1".

In conclusion, SLP implemented with OpenMP offers better performance than the MPI pool algorithm on the benchmark cases. In some sectors of the parallel calculation (Jastrow, $\bar{D}$) must be investigated further in order to understand the poor scaling. As the compilers improves in this area and the number of cores per processor increases the parallel computation should be implement at level of the first loop over electrons. For nodes with more that 10 cores and models with very large number of electrons ($> 10^4$) one should consider nested OpenMP parallelism in order to acquire the best performance.

5 Data input optimisations

We describe briefly the improvements implemented in the input subroutines which read various data sets at the beginning of a computation. The algorithms used are straightforward though laborious to implement.

CASINO versions older than 2.4 uses ASCII input files to store BC data. For system with more that 1000 electrons the reading of this file could take more that 30 minutes which is rather a significant fraction of the maximum allowed run time (12 hours). If the input data file is converted to binary format the reading time drops below 30 seconds. Also an MPI-IO version of this file can be used. This is useful for calculations using MPI-2S algorithm, described in Sec 2, for only one file is needed to store the BC for any value of the number of task per group.

Another significant slowdown, of the order of one 60 minutes, was observed in runs with more that 5000 cores at the reading of the file config.in. Contrary to the BC data file this file is in binary format, the long input time was caused by the fact that in the initial algorithm each task opens the file config.in to read its share of configuration at the beginning of the calculation. The problem was solved by a new version of the configurations reading subroutine in which only a small group of tasks read the file config.in and transfers the data to the associates tasks via MPI communications. In this new version the reading time decrease to tens of seconds even when using $40,000$ tasks.

6 Conclusions

The dCSE project "Improving the parallelisation and adding functionality to the quantum Monte Carlo code CASINO" has achieved its stated goals. CASINO is now capable to compute more complex

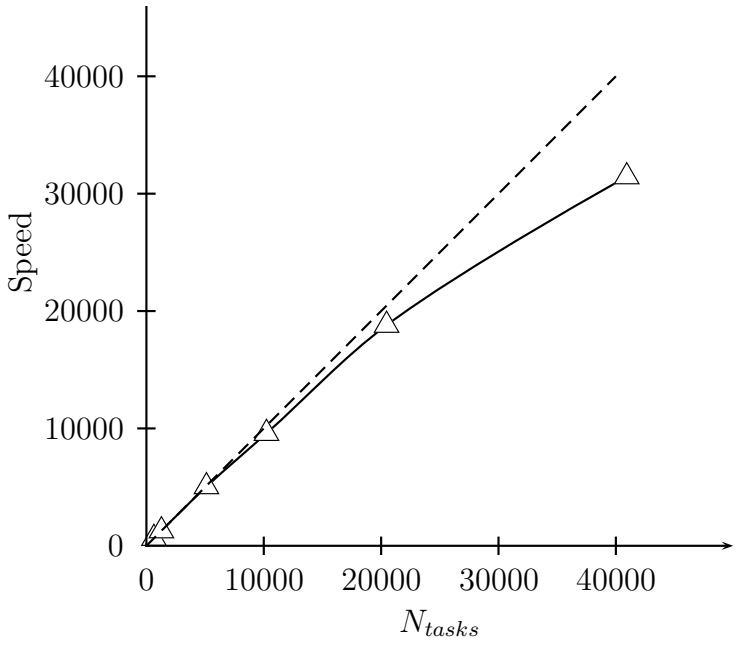

Figure 2: Speed versus number of tasks for a CASINO calculation on Jaguar Cray XT5 at ORNL. The dashed line represents the ideal scaling. Data produced in June 2009.

physical systems with increased efficiency on the latest generation of supercomputers.

We summarise the main improvements implemented in the new version of the code by the dCSE work:

- The System V shared memory solution allows sharing of the OPO data and therefore each core of a processor can run a computing task even for very large scale models. In quantitative terms a computation that needs from 2 to 4 GB of OPO data increases its speed with a factor of 2, with respect to the initial version of the code, and with a factor of 4 if the size the OPO data are larger that 4 GB (for quadcore processors).
- The input optimisations have eliminated unnecessary waiting time for data input which ranged from 30 to 60 minutes saving in this way from 3000 to 6000 AU per 1000 tasks run.
- The computation using mixed mode second level parallelism increases the computation speed with a factor between 1.8–1.6 for systems with a more than 1000 electrons using quadcore processors. The mixed mode parallelism offers the possibility to perform calculations for very large systems (about 10^4 electrons).

The System V shared memory algorithm and the input optimisation have been ported in the release 2.4 of CASINO (July 2009).

The CASINO manual was updated with the necessary information and the Makefile was adapted in order to help users to easily generate the executable that fits their requirements. The second level parallelism is planed to be available in the 2010 release, meanwhile it can be obtained upon request from development branches.

An illustration of the performance level reached by CASINO over the last two years can be seen in Fig 2 which shows that good scaling was obtained for runs up to $40,000$ tasks.

The degradation of scaling for very large number of tasks hints for the need to improve the configuration redistribution algorithm. Another extension which is planned in the next dCSE project is the coupling of CASINO with molecular dynamics codes. This will allow a much faster computation of the QMC corrections to the system energy using the correlated samples technique.

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] The initial code for these functions was provided by David Tanquaray, Cray UK.

[7] Unix man pages.

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

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