# FORTRAN Interface for Code Interoperability in Quantum Chemistry: The Q5Cost Library

S. Borini,[†,‡] A. Monari,*[‡] E. Rossi,[||] A. Tajti,[⊥] C. Angeli,[†] G. L. Bendazzoli,[‡,§] R. Cimiraglia,[†] A. Emerson,[||] S. Evangelisti,[‡] D. Maynau,[‡] J. Sanchez-Marin,[#] and P. G. Szalay[⊥]

Dipartimento di Chimica, Università di Ferrara, V. Borsari 46, I-44100 Ferrara, Italy, Laboratoire de Physique Quantique, Universite Paul Sabatier, 118 R.te de Narbonne, F-31062 Toulouse, France, Dipartimento di Chimica Fisica ed Inorganica, Università di Bologna, V. Risorgimento 4, I-40136 Bologna, Italy, CINECA, Via Magnanelli 6/3, I-40033 Casalecchio di Reno, BO, Italy, Institute of Chemistry, Eötvös Loránd University, P.O. Box 32, H-1518 Budapest, Hungary, and Departament de Química Física, Facultat de Químiques, Institut de Ciència Molecular, Universitat de València, Dr. Moliner, 50, E-46100 Burjassot, Valencia, Spain

Received February 9, 2007

Ab initio quantum-chemistry programs produce and use large amounts of data, which are usually stored on disk in the form of binary files. A FORTRAN library, named Q5Cost, has been designed and implemented in order to allow the storage of these data sets in a special data format built with the HDF5 technology. This data format allows the data to be represented as tree structures and is portable between different platforms and operating systems, making code interoperability and communication much easier. The libraries have been used to build many interfaces among different quantum chemistry codes, and the first scientific applications have been realized. This activity was carried out within the COST in Chemistry D23 project "MetaChem", in the Working Group "A meta-laboratory for code integration in ab initio methods".

## 1. INTRODUCTION

The present article describes the design and setup of a FORTRAN library, Q5Cost, for the management of files containing binary data produced by a generic Quantum Chemistry (QC) program. The data file structure has been specifically defined with the aim of making code interoperability easier in the scientific context of interest of the authors of this work, i.e., quantum chemistry. This activity was carried out within a COST in Chemistry funded project, action D23 "MetaChem"[1,2] and was named "A meta-laboratory for code integration in ab initio methods".

In order to integrate different QC codes in a common work flow, we first have to solve the problem of the different formats adopted by every code in the chain. Our suggestion is to design a "common format for interchange" and to write a converter for each program in the set. Of course, in order not to invent "yet another format", we strongly tried to design a format as general as possible and to coordinate ourselves with other similar initiatives in Europe and elsewhere.

We identified two different kinds of information in QC calculations: small data, mainly ASCII coded, and large data, normally binary coded. While ASCII coded data are described in the proposed format with a specifically designed Mark-up language (QC-ML), large binary data are organized in a HDF5[3] based format, that we called Q5cost. The architecture of the QC-ML format is reported in the first paper of this series,[4] the architecture of the Q5cost data format will be recalled in the next section (and fully described in ref 4), while a description of an XML[5] FORTRAN library (F77xml[6]) is the subject of the third paper of this series.[7]

A specific library was written on top of HDF5 for making the access to the Q5cost data format easier. Since computational chemists, the target users of the library, traditionally use the FORTRAN programming language, the library has been designed to be usable from a FORTRAN 90 program.

This article is organized as follows: In section 2 HDF5, used to store binary data, is described. In section 3, the Q5cost data model is recalled. The Q5Cost library is fully described in section 4. Finally, in section 5, the performance and efficiency of the proposed format are discussed, in section 6 an overview of the interfaces which have been written is presented.

## 2. WHAT IS HDF5 AND WHY IT WAS ADOPTED

The Hierarchical Data Format (HDF) is a general purpose library and file format for storing scientific data. HDF5 was created to address the data management needs of scientists and engineers working in high performance, data intensive computing environments. As a result, the HDF5 library and format emphasize storage and I/O efficiency. For instance, the library is tuned and adapted to read and write data efficiently on parallel computing systems.

HDF5[3] is developed and maintained by NCSA/UIUC. It consists of an abstract model for managing and storing data and a library (with bindings for several programming languages) to implement the data model. The HDF5 library

* Corresponding author e-mail: amonari@fci.unibo.it.
[†] Università di Ferrara.
[‡] Universite Paul Sabatier.
[§] Università di Bologna.
[||] CINECA.
[⊥] Eötvös Loránd University.
[#] Universitat de València.

10.1021/ci7000567 CCC: $37.00 
Published on Web 05/11/2007

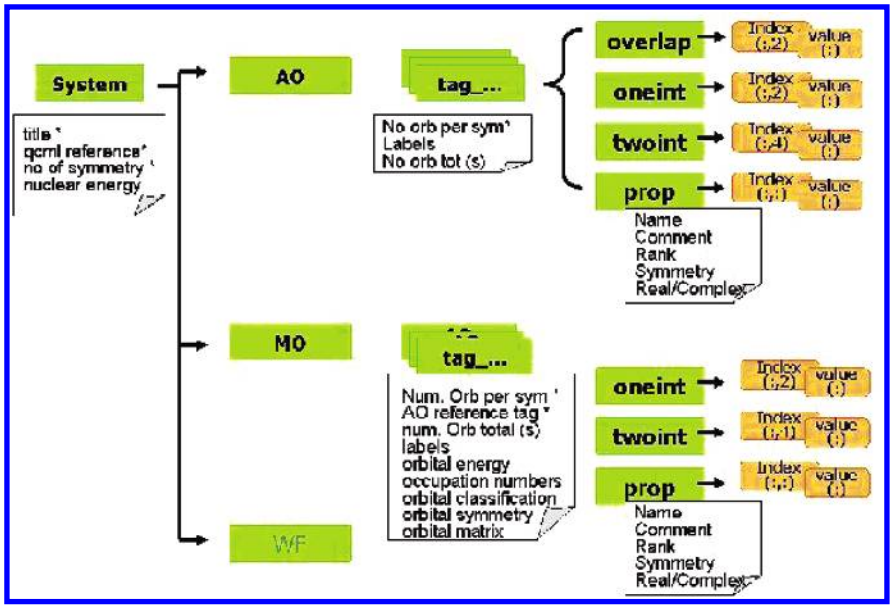


**Figure 1.** The Q5Cost file abstract model.

provides a programming interface to a concrete implementation of the abstract model.

HDF5 can easily handle data described by conventional data structures such as multidimensional arrays of numbers, tables or records, and images, in addition to more complex data structures such as irregular meshes and highly diverse data types. Other important issues are heterogeneous computational environments, parallel data access and processing, the diversity of physical file storage media, and varying notions of the file itself. It also addresses the issues of efficient data access and storage and file portability and supports very large data volumes (practically unlimited). Its flexible data model is extremely useful in multidisciplinary science applications.

Many intrinsic HDF5 features were considered in order to justify our choice, in particular we appreciated the fact that it is a very general data model, allowing great flexibility in the definition of a proper hierarchic data storage schema. Moreover it is very important to recall HDF5 is extensible, portable, and multiplatform (such a characteristic is of great importance considering its possible application in grid environments) and supports an unlimited variety of data type and an unlimited file size. A great consideration has been devoted also to HDF5 performance, since Quantum Chemistry applications usually require very massive and demanding I/O operations.

An HDF5 file has a hierarchical structure and appears to the user as a directed graph, conceptually similar to the UNIX type file system. The nodes of this graph are the higher-level HDF5 objects that are exposed by the HDF5 Application Programming Interfaces (APIs). In particular Groups are the first level containers and, going on with the UNIX file system analogy, can be thought of as directories. Data set, which correspond to files, can be scalars or arrays and actually contain the stored data. Attributes, or metadata, are low-dimensional data (again scalars or arrays) which are used to describe the meaning of the other data.

All the components of an HDF5 file can be easily managed by means of the HDF API. Moreover HDF5 is unique in its ability to physically and conceptually separate data from metadata (attributes), even if they are stored in the same file.

The available HDF5 software tools consist of a number of libraries for each supported programming language (Fortran 90 is one of them) and several utilities for managing data files (inspecting, copying, merging, and so on). It is open source and freely downloadable from the HDF5 Web site.[3]

Using this technology several specific data formats and applications were created in different contexts. A wide list of tools (both commercial and open source) based on HDF5 can be found on the Web.

HDF5 is a widely used tool for scientific data representation, and its characteristic of flexibility, hierarchic structure, portability, and performance make it, at our advice, the ideal candidate upon which a common format for the storage of binary, large dimension quantum chemistry data can be built (http://hdf.ncsa.uiuc.edu/whatishdf5.html).

## 3. THE Q5COST FORMAT ARCHITECTURE

The structure of Q5cost data format has been fully described elsewhere.[4] Here we just want to briefly recall its main features.

Q5cost is a common interchange format for large binary data coming from quantum chemistry calculations. It is flexible and extensible, so we can design it in an incremental way. The data format consists of a collection of chemical objects related within a hierarchical structure in a logical containment relationship as reported in Figure 1. The green boxes are containers (groups), and the yellow boxes are data (data sets), while the metadata (attributes describing the data) are listed in the white labels.

**The Data Model.** This extensible data model has been constructed using as a basis the experience of the research groups involved. After a preliminary analysis, some clear points have emerged.

The first point is that many different types of simple and small data must be handled (nuclear energy, molecular orbital labels, molecular symmetry, and so on). We will refer to these data as "metadata", in order to distinguish them from the real large information on the chemical system such as the integral values.

Metadata represent well-known chemical entities and belong to three generic data classes: scalars, vectors, and matrices. For example, the nuclear repulsion energy is a floating point scalar, molecular orbitals are an (N,M) floating point matrix, the associated orbital energies are a floating point vector, the molecular orbital labels are a vector of strings, and so on. The library should provide an interface for accessing these data both as generic or specialized entities.

A second point is that in quantum chemistry large matrices with an arbitrary number of indices (rank-n arrays) are very common data structures. These data usually scale aggressively with the system size, and they are normally accessed with a "chunked" approach (i.e., using well-defined blocks of data). This is not only the case of entities like two-electron integrals, or atomic orbital overlap, but also other more application-specific information, like the four particle density matrix. In the modern approaches based on localized orbitals, these matrices are of sparse nature; this encourages the storage of only nonzero elements, each one associated to *n* indices in the case of a rank-*n* array. This representation of the data, although not particularly efficient in terms of space occupation, is well-known by the interested parties, easy to debug, and already integrated in the current code-base both for memory representation and file storage of data.

These large array data share common features:

- They are usually integrals, whose evaluation involves one or more operators and a given (large) number of functions. These functions are referenced by the indices of the matrix. For example, two-electron integrals on the molecular orbital basis are stored as a rank-4 array with indices referring to the molecular orbitals; in the case of atomic basis set overlap integrals, the indices refer to the atomic basis set orbitals.
- The rank of the matrix depends on the physical meaning of the integrals. The atomic basis set overlap is described by two indices and can be stored as a rank-2 array, two-electron integrals have four indices imposing a rank-4 array, and the four particle density matrix has eight indices, requiring a rank-8 array.
- Additional information is needed to describe the operator involved (for example its symmetry) and the entities the indices refer to.

Thus, all these data objects can be described by a "generic property" object, provided that we define the matrix rank and the involved operator(s) and basis functions.

Since some of these "properties" are well-known chemical entities, and chemists are used to referring to them by name, we have planned a specific library access to most of them (overlap, one-electron integrals, two-electron integrals...), in addition to an interface to the "generic property" for handling other properties not explicitly provided by the library. This should ensure both ease of use and general adaptability of the library to either alternative or future theoretical developments.

The last point is that all these chemical objects are related within a hierarchical structure, and logical containment relations can be defined for them.

A first (root) container, named System, represents the molecular system as defined by its structural data (chemical composition and spatial geometry). To this container can be associated all the metadata that are invariant at the level. Since data about the chemical composition, the geometry, the basis set. and the symmetry are contained elsewhere (in the QC-ML file), here the only relevant metadata remain the nuclear repulsion energy and information on the symmetry group, needed for accessing the property data.

A system can contain several "Domains". The role of the Domain is to group together Property entities whose indices conceptually refer to the same kind of functions. Three Domains have been recognized as fundamental: AO for Atomic Orbital, MO for Molecular Orbitals, and WF for Wave Function.

The AO Domain holds properties referring to the atomic basis set functions: overlap, one-electron, and two-electron integrals on the atomic basis set, in addition to the generic property. The invariant metadata consist of information on the atomic orbitals, like their number, the labels, and symmetry.

The MO Domain holds properties referring to molecular orbitals: one-electron and two-electron integrals on the MO basis set, in addition to the generic property. The descriptive metadata for the domain refer to the MO basis description: their number, labels, and symmetry, the AO basis they were derived from, the matrix collecting the coefficients of the MO expansion on the AO basis, orbital energies, classification, and occupation numbers.

The WF Domain holds properties referring to the electronic states. The complete definition of this container is not available yet. It is still the subject of research and development, given its noncritical nature for the first deployment and test of the library.

For each of the domains, different occurrences can be defined by means of an identifier (tag) chosen by the user, with a default value if no tag is provided. The aim is to provide storage for multiple entries, like in the case of multiple molecular orbitals in the MO Domain or multiple basis sets in the AO Domain.

The bottom level of the hierarchical scheme is made of the properties. Even if from the user's point of view many different "properties" are available, all of them are different instances of the same "generic property" object. This object holds the true data, i.e., the integral values and the corresponding index values. Also here, in order to fully define the nature of the actual property, some metadata are needed: name, rank, symmetry, and type (i.e., real, imaginary, or complex).

In ab initio codes, the two-electron integrals, either on the atomic or the molecular basis set, are among the largest data set. For this reason, an efficient management of these integrals is crucial for obtaining a good performance. The whole set of *N* integrals, with the corresponding indices, can be stored within a linear structure like that reported below

$$(\mathrm{val}_1;\, i_1, j_1, k_1, l_1)$$

...

$$(\mathrm{val}_N;\, i_N, j_N, k_N, l_N)$$

where val is the floating point integral value, and *i*, *j*, *k*, and *l* are the corresponding integer indices. The simplest solution is to store both the integrals and the four indices, so the order of the records does not matter. Moreover, null or small

integrals can be simply omitted from the list, a fact particularly important when working with local orbitals. For this reason, at the moment, this is the only strategy that was adopted in the Q5cost data format. The price one has to pay is the additional storage of the four integer orbital labels, leading to a memory/disk occupation that could be three times larger than the one if only integrals were stored (in the common case of REAL*8 integrals and INTEGER*4 indices). In the case of very large integral files, this overhead can be extremely heavy. For this reason, in many QC programs the integrals are stored in a well defined order, the "standard order", so that the orbital labels can be omitted without loss of information. (In the presence of spatial symmetry, a large number of zero integrals are present, and the standard order can be modified in order to take this fact into account).

At present only the simplest solution has been implemented in Q5cost. Of course we are aware that, for the sake of generality, it is important to provide for the possibility to store integrals also in the other way, allowing the choice among one or more definite orders.

## 4. THE Q5COST LIBRARY

The Q5Cost library provides read and write access to files defined in accordance with the data model described before (Q5cost data model). It provides a specifically designed high-level access for quantum chemistry developers. The rationale is to provide a FORTRAN interface based on well-known chemical entities, rather than groups or data sets like in the original HDF5 interface. HDF5 takes care of the low level management of the file, and Q5Cost provides the high-level Application Programmer Interface for storage and retrieval of chemical entities.

**Library Structure.** The library is written in FORTRAN 95 and consists of several modules, each one providing different facilities. The most important modules are as follows:

- **Q5Cost**: defines the high-level API. This module provides subroutines designed to be at the disposal of the final programmer.
- **Q5Core**: provides a wrapping facility for HDF5 routines, in order to perform additional useful services like reference counting and debugging. It also provides simplified routines to perform frequently used low-level tasks.
- **Q5Error**: provides facilities for high level debugging of library and client codes. This module implements a ring buffer for error messages, different logging levels, generic reference counting for catching memory leaks, and a subroutine call stack trace.

The names of the subroutines in each module are identified by an appropriate prefix and have been chosen to provide an explicit and intention revealing interface to the entities described in the previous section. Although FORTRAN 95 does not allow object oriented (OO) programming, some OO concepts have been used in the development of the library but taking into account the possible procedural programming background of future developers. The state is preserved in the HDF5 file, and subroutines refer to the file directly through the HDF5 file identifier, an easier concept for FORTRAN programmers more used to file descriptors.

**The Q5Cost Module.** This module is the main reference for the final user. It provides subroutines to read and write HDF5 files in the Q5cost format with a high level of abstraction. Using this library the users can deal with high level concepts without worrying about low level implementation details. If a finer access is required for the underlying HDF5 file, the Q5Core module provides this type of access in a simpler way with respect to the raw HDF5 routines.

All the routines in the Q5Cost module have the Q5Cost_ prefix, and they are organized in several classes:

- **Init**: initialize and deinitialize the library within the program.
- **File**: create, open, close the .q5 file, and write/get root attributes, like creation time, access time, and file version.
- **System**: create or check the existence of the System and set/get the specific attributes
- **AO**: create or check the existence of a given occurrence of the AO Domain and set/get its attributes
- **AOOverlap**: create the folder, read and write data for the atomic basis set overlap property
- **AOOneInt**: create the folder, read and write data for the one-electron integrals in atomic aorbitals basis
- **AOTwoInt**: create the folder, read and write the data for the two-electron integrals in atomic orbitals basis
- **MO**: create or check the existence of a given occurrence of the MO Domain and set/get its attributes
- **MOOneInt**: create the folder, read and write data for the one-electron integrals in molecular orbitals basis
- **MOTwoInt**: create the folder, read and write the data for the two-electron integrals in molecular orbitals basis
- **WF**: create or check the existence of the "WF" domain and set/get its attributes
- **Property**: create the folder, read and write data for a generic property. The name, domain, rank, and type have to be defined by the user.

Additional routines are available for the generic access to the "Property" class, allowing the management of user defined properties. Subroutines like AOOverlap, MOOneInt, and MOTwoInt contain calls to these property routines, passing the specific parameters of the involved property.

The routines of the Q5Cost module provide a context-based access to chemical entities. This access is converted into a path-based access, creating an appropriate layout for HDF5 groups, data sets, and attributes, and writing the user provided data into the file. Some data are provided automatically by the library, like the creation or access time and the Q5Cost library version.

One important aspect of this format is that the user is not forced to enter all the quantities; he can store the quantities that are actually available, or in which he is interested, and add other data later when available. Constraint checks are however mandatory in order to ensure basic file consistency. For example, a MO Domain can be created only if a System and an AO Domain exist, in order to guarantee the presence of fundamental data, like the number of symmetry species and the number of basis functions for each symmetry species.

**The Q5Core Module.** The Q5Core module is a low level module designed to provide wrapping facilities between HDF5 and Q5Cost. At the moment it is focused on providing additional debug information, reference counting for HDF5 objects, additional low-level API for simplifying common tasks, and so on. This module provides path-based manage-

```
-------------

!  Initialize Q5Cost and opens Q5 file
    call Q5Cost_init(error)
    call Q5Cost_file_open(filename,file_id,error)
    call Q5Cost_System_get_num_sym(file_id,num_sym,error)!get order of the symmetry group
    allocate (num_orb_symm(num_sym)) !Allocate array containing orbitals for symmetry classes
    call Q5Cost_MO_get_num_orb_sym(file_id,num_orb_symm,error)! Get the array of orbitals for
symmetry classes
    call Q5Cost_System_get_nuc_energy(file_id,nuc_rep,error)! Get nuclear energy
    call Q5Cost_MO_get_num_orb_tot(file_id,num_orb,error)! Get number of orbitals
------------
! Actually reads two electron integrals
   KOUNT_MDMO=0
   KOUNT_BI=0
   offset=0
   howmany=chunk
   howmany_fixed=chunk
   DO
! Acctually reads integrals
     call Q5Cost_MOTwoInt_read(file_id,offset,howmany,idx_bi,value_bi,error)
     offset=offset+howmany
     KOUNT_BI=KOUNT_BI+howmany
     IF (key_word .ne. "bin") THEN
       DO II = 1,howmany
! Actually writes integrals
          WRITE(10,'(1X,D20.13,4I4)') value_bi(II),idx_bi(II,1),idx_bi(II,2),idx_bi(II,3),idx_bi
(II,4)
       ENDDO
     ELSE
       DO II = 1,howmany
! Actually writes integrals in binary file
          WRITE(20) value_bi(II)
       ENDDO

    IF (howmany .lt. howmany_fixed) EXIT
     howmany=howmany_fixed
   ENDDO
-----------------------------------------------------
```

**Figure 2.** Q5cost library using example.

ment of scalar, vector, and matrix entities (in contrast with the context-based approach of the Q5Cost module, which focuses on chemical concepts rather than HDF5 path). It also provides routines for the easy handling of the Property data (indices and values), relative to a CompactMatrix class (CM). End users in general should not access Q5Core module routines.

The Q5Core module guarantees the transparency of the Q5cost data model with respect to the underlying technology. In case we decide to use another storage format in place of HDF5, only this module should be modified. The Q5Cost module, i.e., the end user interface, remains unchanged, being independent of the low-level format.

**The Q5Error Module.** The Q5Error module provides subroutines for debugging and monitoring the behavior of the library and the application code. A ring buffer is provided to keep track of error messages generated by the library. A verbosity level can be set, from totally silent to highly verbose; in the latter case each subroutine call and return is reported in the buffer. Moreover, a stack for backtracking has been implemented to keep track of the call tree. The tree is printed out when an error occurs or when error reporting is requested. Different specific error codes have been provided for, to report anomalous behavior of the application code or of the library itself. The error codes are defined as numeric parameters and report situations ranging from invalid parameters to nonexistence of some information in the file. The presence of an error condition is returned to the application code through the last parameter of each subroutine.

**Additional Details: Test Suite and Documentation.** A test suite has been designed and implemented in order to verify the library correctness in a high number of well-known critical situations. At present, more than 250 tests are available, covering most common usage patterns and performing reference counting to prevent leaks of HDF5 references. The test suite provides an effective tool for debugging and bug fixing.

Library documentation is embedded into the FORTRAN code as comments, using a custom tag system to provide meta information about each comment. A simple parser, written in the PYTHON programming language, extracts the documentation producing HTML files.

**Coding Examples.** For the user convenience we would like to introduce here a very simple coding example demonstrating the very user-friendly nature of the Q5cost library and its strong relation with quantum chemical objects. If, for example, one wants to read some integrals from a Q5Cost file (a very common task since many QC codes rely on other programs for the computation of basis orbital integrals) the programers should use the model code example given in Figure 2.

In this particular example it is clearly evidenced how the simple call to Q5Cost_MOTwoInt_read allows for the reading of a chunk of integrals (together with their indices) of size howmany. The value of howmany being both an input and an output value for the routine, if the howmany value is changed on output, then we are at the end of the data set, and therefore we read all the integrals. Similar coding is required to write integrals or accessing metadata.

## 5. PERFORMANCE AND EFFICIENCY ASSESSMENT

As we have already discussed, the Q5cost format was intended as a file exchange format between different platforms and codes and not as an internal format to be used during actual computations. For this reason, performance considerations have been considered to be less important than other features such as transparency of file format or code and file portability. But to ensure that the library does not impose excessive overheads in terms of CPU time or disk space, we decided to undertake some comparisons with ordinary binary files.

**Table 1.** Writing Time versus Chunk Size[a]

| buffer size | time (s) Fortran binary | time (s) Q5cost | buffer size | time (s) Fortran binary | time (s) Q5cost |
|---|---|---|---|---|---|
| 1024 | 265.23 | 226.62 | 16 384 | 18.86 | 17.04 |
| 2048 | 121.13 | 114.53 | 32 768 | 8.56 | 6.09 |
| 4096 | 62.38 | 59.02 | 131 072 | 6.19 | 4.86 |
| 8192 | 34.39 | 31.46 | 262 144 | 5.84 | 4.08 |

[a] Number of integrals 15 000 064, binary file size 343 Mb, .q5 file size 346 Mb.

**Table 2.** Space Occupation and Writing Time versus Number of Integrals for a Fixed Chunk Size of 16 384 Integrals

| | Q5cost | | Fortran binary | |
|---|---|---|---|---|
| number of integrals | size | write time (s) | size | write time (s) |
| 16 384 | 397 Kb | $5.00 \times 10^{-2}$ | 384 Kb | $5.00 \times 10^{-2}$ |
| 65 536 | 1.5 Mb | $1.00 \times 10^{-1}$ | 1.5 Mb | $1.00 \times 10^{-1}$ |
| 114 688 | 2.7 Mb | 0.15 | 2.6 Mb | 0.17 |
| 507 904 | 12.0 Mb | 0.62 | 12 Mb | 0.68 |
| 1 015 808 | 23.0 Mb | 1.21 | 23 Mb | 1.37 |
| 5 013 504 | 115.0 Mb | 5.88 | 115 Mb | 6.41 |
| 10 010 624 | 231.0 Mb | 11.11 | 229 Mb | 12.12 |
| 50 003 968 | 1.1 Gb | 56.19 | 1.1 Gb | 64.21 |
| 100 007 936 | 2.3 Gb | 125.32 | 2.2 Gb | 148.53 |

All performance tests have been run on a single node of an IBM Linux Cluster 1350 at CINECA (Intel Xeon Pentium IV, 3 GHz 512 Cache, equipped with a GPFS storage disk). The software was compiled with the Intel FORTRAN Compiler 8.1 and run under Suse Linux SLES 8.

In order to perform the tests we wrote a specifically designed code that

- Creates a proper.q5 file with its internal structure (System, AO, MO..)
- Opens a normal binary file
- Writes in the .q5 file a number of two-electron integrals specified by the user with the proper format: a one-dimensional array of reals (values) and a four-dimensional array of integers (indices) using a chunk whose size has been specified by the user.
- Writes the same number of two-electron integrals in a binary file together with the four indices. For this operation a buffer of the same size of the chunk specified previously is used.
- Computes the time necessary to write the .q5 and binary files and calculates their sizes.

In the first test we evaluated the time needed to write a file of approximately 300 Mb, using different chunk sizes; the results are reported in Table 1.

As it can be seen the time needed to write the .q5 file is less than the time needed to write the ordinary binary file for any chunk size. This feature is a direct consequence of the use of the HDF5 library, whose performance characteristics are well documented.[3,8] Obviously using chunks of 1 GByte size, hence limiting the number of accesses to the file, decreases rapidly the time needed for the entire process.

In the second test we studied the time needed to write .q5 and binary files with a fixed chunk size (16 384) and the corresponding size of the file so produced. The results are collected in Table 2. It can be seen that the sizes of the .q5 files are comparable with the binary file sizes. The .q5 files are, in fact, only bigger by less than 1% compared with the ordinary binary files.

The main problem regarding disk occupation is that all four indices are stored for two-electron integrals and this leads to large file sizes. It is possible to avoid storing the indices by using a predefined order; we are currently working on implementing such a mechanism in our library.

## 6. HOW THE LIBRARIES HAVE BEEN USED: THE INTERFACES

The Q5Cost library has been used to write interface programs (wrappers) for converting data from the output of several ab initio programs to the Q5cost data format and vice versa.

A detailed description of the wrappers developed until now has been reported elsewhere.[4] Here we will present a simple overview to help the reader understand the capabilities of the Q5Cost libraries.

In general, a wrapper should accomplish a quite simple goal: read quantities stored in a given data format and write them in a different data format using (when available) the specific I/O library for the two formats.

Two zero-level programs (Columbus[9] and Dalton[10]) have been fully integrated, and their integrals can actually be converted into the Q5cost format by a specific wrapper.

The wrapper for the Molcas[11] code is under development: we plan to integrate it in the official Molcas distribution by using the "module" concept of Molcas.

Other proprietary codes, developed by some of the partners of the project, have been fully integrated in the Q5cost format, either directly or via the use of wrappers.

In particular the Bologna FCI[12] code can directly read Q5cost files, while the Toulouse code chain (especially CASDI[13] and EPCISO[14]) can access Q5cost data through a specific wrapper.

Thanks to those first wrappers we had the opportunity of connecting programs, not able to communicate before, in a single workflow. A preliminary application was the study of dispersion interactions on the neon dimer that was carried out within this framework and gave satisfactory results.[15]

## 7. CONCLUSIONS

A FORTRAN library, based on HDF5, has been realized for the management of Quantum Chemistry data in the Q5cost data model. The library is intended to give an easy high level access to the data model based on chemical concepts.

The library is based on HDF5 technology and inherits its best features (performance, logical structure, multiplatform portability) while offering to chemists a user-friendly FORTRAN interface. Using the library, a number of wrappers have been written for exchanging information among different, often complementary, QC programs. The first scientific applications have been performed too, showing the actual usability of the tool.

## REFERENCES AND NOTES

(1) COST in Chemistry Action D23. http://costchemistry.epfl.ch/ (accessed April 2, 2007).

(2) Rossi, E.; Emerson, A.; Evangelisti, S. *Lect. Notes Comput. Sci.* **2003**, *2658*, 316−323.
(3) HDF5 a general purpose library and file format for storing scientific data. http://hdf.ncsa.uiuc.edu/HDF5/ (accessed April 2, 2007)
(4) Angeli, C.; Bendazzoli, G. L.; Borini, S.; Cimiraglia, R.; Emerson, A.; Evangelisti, S.; Maynau, D.; Monari, A.; Rossi, E.; Sanchez-Marin, J.; Szalay, P.; Tajti, A. A common data format for Quantum chemistry codes. *Int. J. Quantum. Chem.* In press.
(5) Holmer, A. *XML IE5 − Programmere's Reference*; Wrox Press: Chicago, IL, U.S.A., 1999.
(6) Borini, S. f77xml. http://members.ferrara.linux.it/munehiro/f77xml/ (accessed April 2, 2007).
(7) Borini, S.; Evangelisti, S.; Monari, A.; Rossi, E. A FORTRAN interface for code interoperability in Quantum Chemistry: The F77/F90xml library. To be submitted for publication.
(8) Yang, M.; McGrath, R. E.; Folk, M. Performance Study of HDF5-WRF IO, WRF Workshop, June 26, 2004.
(9) Lischka, H.; Shepard, R.; Shavitt, I.; Pitzer, R. M.; Dallos, M.; Müller, Th.; Szalay, P. G.; Brown, F. B.; Ahlrichs, R.; Böhm, H. J.; Chang, A.; Comeau, D. C.; Gdanitz, R.; Dachsel, H.; Ehrhardt, C.; Ernzerhof, M.; Höchtl, P.; Irle, S.; Kedziora, G.; Kovar, T.; Parasuk, V.; Pepper, M. J. M.; Scharf, P.; Schiffer, H.; Schindler, M.; Schüler, M.; Seth, M.; Stahlberg, E. A.; Zhao, J.-G.; Yabushita, S.; Zhang, Z. *COLUMBUS, an ab initio electronic structure program, release 5.8*; 2001.
(10) *DALTON, a molecular electronic structure program, Release 2.0*; 2005. See http://www.kjemi.uio.no/software/dalton/dalton.html (accessed April 2, 2007).
(11) Andersson, K.; Barysz, M.; Bernhardsson, A.; Blomberg, M. R. A.; Carissan, Y.; Cooper, D. L.; Fülscher, M. P.; Gagliardi, L.; de Graaf, C.; Hagberg, D.; Hess, B. A.;. Karlström, G.; Lindh, R.; Malmqvist, P.-Å.; Nakajima, T.; Neogrády, P.; Olsen, J.; Raab, J.; Roos, B. O.; Ryde, U.; Schimmelpfennig, B.; Schütz, M.; Seijo, L.; Serrano-Andrés, L.; Siegbahn, P. E. M.; Stålring, J.; Thorsteinsson, T.; Veryazov, V.; Widmark, P.-O. *Molcas, version 6.2*; University of Lund: Sweden.
(12) Bendazzoli, G. L.; Evangelisti, S. *J. Chem. Phys.* **1993**, *98*, 3141.
(13) Maynau, D.; Evangelisti, S.; Guihery, N.; Calzado, C. J.; Malrieau, J. P. *J. Chem. Phys.* **2002**, *116*, 10060−10068.
(14) Vallet, V.; Maron, L.; Teichteil, C.; Flament, J. P. *J. Chem. Phys.* **2000**, *113*, 1391−1402.
(15) Monari, A.; Bendazzoli, G. L.; Evangelisti, S.; Angeli, C.; Borini, S.; Maynau, D.; Rossi, E. The effect of the Basis-Set Superposition Error on the calculation of dispersion interactions: A test study on the Neon dimer. accepted by the *J. Chem. Theor. Comput.*

CI7000567