Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVI, pp.455-467
Sergio A. Elaskar, Elvio A. Pilotta, Germán A. Torres (Eds.)
Córdoba, Argentina, Octubre 2007

HIGH PERFORMANCE COMPUTING USING SERIAL APPLICATIONS: EXPLOITING TOOLS OF THE QUEUE SYSTEM

M. F. Ciappina[a,b] and W. R. Cravero[b]

[a]*Max Planck Institute for the Physics of Complex Systems, Nöthnizer Str. 38, D-01187, Dresden, Germany, ciappi@pks.mpg.de*
[b]*CONICET and Departamento de Física, Universidad Nacional del Sur, Av. Alem 1253, B8000CPB, Bahía Blanca, Argentina, wcravero@uns.edu.ar*

Keywords: Parallel programming tools, queue systems, computational time optimization.

Abstract. During the last years we have been witnesses to an impressive growth in the calculation power of computational systems. Furthermore, software tools and programming languages have followed these advances providing new techniques and programming paradigms to use the installed hardware in more efficient ways, including the use of parallel programming. Unfortunately, to take full advantage of the power of these computational systems it is necessary, in most cases, to rewrite and optimize our older routines and, more often than not, to attack the problem again from scratch. Nowadays it is usual for many people working in research institutions to have access to a cluster of computers managed by an intelligent queue system. This tool allows to manage the load and the different users priorities efficiently. In this work we show how it is possible to continue using our old and reliable routines, written in any language, and, at the same time, to take advantage of a simple distribution technique to reduce dramatically the computational time. We will show several examples where we have applied this technique emphasizing the fact that it is not necessary to modify substantially our original programs. These examples comprise different problems in collision physics, some of them including Monte Carlo approaches, and processes in laser-atom and molecule interactions, in which it is necessary to deal with Fourier Transforms techniques.

1 INTRODUCTION

The growth in the computer systems has fulfilled, or in some cases exceeded, the well-known Law of Moore, who established the duplication of the numbers of transistors integrated in a processor, which is closely related to the computational power itself, every 24 months (Moore (1965)). The evolution of the programming languages has been also quite evident. On these new languages innumerable tools have been developed and implemented both in the so-called Free Software and within the framework of proprietary one. Since the upcoming of PC clusters managed by Linux operating system, the parallelization tools have been profusely developed and nowadays are being used intensively in these systems. Nevertheless, a great number of algorithms and routines created in the past to solve specific problems, have been written and are still being used in serial form. In order to parallelize these piece of codes and to use them in an efficient way in clusters, it is necessary to rewrite them *from scratch* or to perform a great amount of changes. This may invalidate already certified codes or may simply not be technically or economically viable. Let us mention as an example, the countless routines in FORTRAN language (77 or 90) that have proved largely their effectiveness and reliability, that cannot take full advantage of the computational power of the parallel systems, since they were built in an serial programming environment.

There exists a great number of problems in which it is necessary to use the same program or routine hundreds or thousands of times, only changing one or several of the entrance parameters. Among these problems we can mention the calculation of differential cross sections within the framework of atomic collisions. These magnitudes are the result of experiments performed in the atomic physics facilities around the world. From a theoretical point of view, the computation and modeling of differential cross sections imply a great number of calculations, usually involving the computation of special functions and the use of multidimensional integration. With the impressive advance of the experimental techniques, it has been necessary to refine even more the theoretical approaches and to make many tests with numerous parameters. It is for that reason that all the saving in computational time that can be obtained is welcome. Since the collision theory has 5 or 6 decades of constant development, naturally an enormous number of routines and programs have been developed. Many of them have largely proved their efficacy and reliability and deserve to be reconsidered in this new context of parallel environments (for a recent review of atomic and molecular physics see e.g. Drake (2006) and references therein).

In this work we will describe how to use our old programs and routines in a concurrent way using tools that the software installed for cluster management and balance brings us, under Linux operating system. It is necessary to briefly recall how the environment in this kind of systems works, how it sends programs to run, what is a queue and how it works and different commands that we use along our work. Although we will apply our technique to problems within the framework of atomic and molecular physics, we are confident that it is perfectly possible to extend our *recipes* directly to a number of diverse problems both in other physical branches and computer science.

2 GRIDS AND SOFTWARE FOR CLUSTER MANAGEMENT

One can think a Grid as a collection of computational resources prepared to perform diverse tasks, among them mathematical and/or scientific calculations. In its simpler form, a Grid appear as a huge computing system that provides an unique access point to these distributed systems. For the users, in most of the cases, it is *transparent* to perform tasks in this type of systems. Furthermore, the efficient management of the installed resources in this kind of

huge and heterogeneous systems is a very complex task. Fortunately, there exists tools that are installed *above* the operating system and will allow us to alleviate enormously these tasks. Among them we can mention the software for management of grids and clusters known as Sun Grid Engine (SGE), in its different flavors (Sun Microsystems Inc. (2002)). This tool, of which we will describe some commands that we use throughout the work, perform the balance and management of users in large computing systems and clusters under Unix/Linux operating systems.

The users policies and load balance of a large computing system are of fundamental importance, since they will allow us to optimize and to take full advantage of the whole computational resources installed. With SGE the users can send to execute thousand of programs without worrying how these programs are being executed and which type of resources they are using. Since we use SGE at user level, not administrator, we will not describe here the details about the versatility that this tool has to perform the load balance and the management of resources and policy of priorities. We will only mention that SGE provides many commands and tools to construct different users policy and allow the management of the computational resources tailored for each system and administrator.

SGE has numerous commands at user level, but we will use and describe in detail only a couple of them, emphasizing some of their most important options. These commands will be important for the execution purposes in which we are interested. Let us explain in a few words what we call here *jobs* and which is the meaning of *queues* in this context. We can do an analogy between a Grid and the queue of a check–in in an airport. The *jobs* correspond to the passengers. *Jobs* waiting to be executed in a specific processor or node would be the passengers who are waiting for attention. The queues, which provide services for the *jobs*, are formed by the airlines employees. As in the case of passengers, the requirements of each *job*, such as memory, execution time, processor type, software licenses, among others, can be very different. Consequently only certain queues are able to provide the necessary resources for each *job*.

Following such analogy, the SGE arbitrates the available resources and the requirements of each *job* in the following way: a user sends a given *job* for execution and SGE obtains the profile of requirements of it (memory, execution time, special architecture of the processor, requirements of i/o and software, among others) from the user profile or from some definition by default. Furthermore, the system obtains the user identity and stores the time at which the *job* was sent for execution. Once a particular queue is available to execute a new *job*, the SGE determines if such queue fulfill the *job* requirements. If such *matching* is successful, SGE sends immediately the *jobs* of greater priority or those than have had the larger delay time. The queues allow us to concurrent execution of several *jobs*, but the optimization of this task will be one of the task of the administrators, who also will decide about specific users and priority policies.

Throughout the *timelife*, each *job* will be associated with a specific queue, that in the simplest case will correspond to a unique processor in a certain node. If for some reason a queue is suspended, all the associated jobs with it also will be suspended. Although, it is possible to explicitly specify the queue in which *jobs* will be executed, SGE automatically deal with this task according to the *job* profile and to the availability of resources in each queue. We note if we sent explicitly for execution a *job* to a determined queue, it will remain *linked* to that queue and as a result the SGE daemons will not be able to select the less loaded node with the subsequent loss of performance and delays in the execution time.

Next, we describe how to send for execution a program and which are the most important parameters in our context. The command to perform this last task is `qsub` and the syntax is the following:

```
qsub job.sh
```

This command assume that `job.sh` is the name of a *script file* and that such file is placed in the corresponding home directory. The `qsub` command will confirm the successful sending of the *job* in the following way:

```
your job 1 (‘‘job.sh’’)
has been submitted
```

We can obtain information about the state of our *job* doing

```
qstat -u username
```

where `username` is our user name in this environment. For each *job* the status report will produce a list including the following items:

- Job ID, which represents a unique number that is included in the sending confirmation
- *job script* Name
- *job* owner
- Status indicator; for example `r` means running and `t` targeted to an specific queue and ready to be executed
- Sending or start date and time
- Queue name in which *job* is being executed

Let us list the options of the command `qsub`. One option that will be useful for us is

```
qsub -t N1-N2 job.sh
```

where `N1` and `N2` are natural numbers. Even when `N1` and `N2` can take any value, for simplicity reason, and without loss of generality, we will only consider the case `N1=1` and `N2=N`. The `qsub` command with this option will send to run an *array* of `N` copies of the *script file* `job.sh`. We can think that this option is of limited or none utility, since we are sending for execution exactly the same *script file*, i.e `job.sh`, `N` times. However, let us discuss how we can take advantages of this scheme. First, SGE has associated for each *array* an index which is saved in the variable `$SGE_TASK_ID`. This variable will take values between 1 and `N`, since we have `N` elements in our *array* of *script files*.

The second point to take into account is the possibility to redirect the standard input (keyboard) and output (screen) that the Linux operating system, based in older Unix environments, bring us. That is, we know that with the commands < and > we can change the standard input and output devices, i.e. the keyboard and the screen, respectively, to a *physical* input and output data files.

In the following example we will show in a *script file* how the `-t 1-N` option works together with the input and output redirection. Let us consider the following *script file* called `job.sh`

```
1  #!/bin/bash
2  #$ -o $HOME/wd/
3  #$ -e $HOME/wd/
4  #$ -l h_rss=1500M,h_fsize=3000M,
h_cpu=900:00:00,s_cpu=899:59:00,hw=x86_64
5  $HOME/wd/job.exe < $HOME/wd/data$SGE_TASK_ID.in
> $HOME/wd/data$SGE_TASK_ID.out
```

The first line describes the type of shell in which we will execute the *script file*, in this case the `bash` shell. Lines 2 and 3 correspond to specific directives used by the `qsub` command. The `-o` (`-e`) option redirect the output (errors) messages to the `$HOME/wd/` directory. Line 4 contains directives related with the *job* profile (see the discussion above in this section) and specify the memory requirements, execution time and processor architecture needed for the correct execution of the program `job.exe` (see Sun Microsystems Inc. (2002) for more details).

The most important line in our discussion is the number 5. We see that our program `job.exe` receives as input the files `data$SGE_TASK_ID.in`,

```
< $HOME/wd/data$SGE_TASK_ID.in
```

that will be different since `$SGE_TASK_ID` change for each element of the array. These files contain the different parameters that allow us to execute concurrently the program `job.exe` for different cases (we will show examples in the next Section).

As the results obtained from `job.exe` will be different, since we are using different input parameters, we should to save the results in different data files. This last task can be done with

```
> $HOME/wd/data$SGE_TASK_ID.out
```

One important point to be mention is that in our program `job.exe` the input and output devices, that will be parallelized after, should be put by default (keyboard and screen respectively). For example, in FORTRAN 77 or 90 this task can be performed with the commands

```
READ(*,*) V
READ(*,*) ANGLE
```

to read, e.g., the variables `V` and `ANGLE` and using

```
WRITE(*,*) FDCS,ENERGY
```

to write the variables `FDCS` and `ENERGY`. Afterward the standard input and output devices will be redirected to data files, using the procedure we have described above, with the commands `<` and `>` respectively.

We further note that both the redirection and the use of the SGE commands are independent of the programming language in which our routines and programs were written and compiled.

2.1 Scalability

We can not consider scalable all the physics problems that are able to attack and model computationally. Consequently, there exist problems, for example those associated with the temporal evolution of physical systems, where the parallelization and scalability is particularly complex. This difficulty is related to the fact that, in the most of the time–dependent cases, is necessary to know the result of older time steps to perform calculations in the present temporal

one. On the other hand, there exists numerous problems where the scalability is trivial, in the sense that is necessary to execute the same routine or program hundreds or thousand of times, with different entrance parameters. We will describe in detail several examples this last kind of problems in the next Section. Furthermore, it bears mentioning that along the examples we will use our older serials routines, only with slightly modifications. We show how is possible to increase substantially our productivity and we discuss the implications of this.

3 APPLICATIONS

In the last Section we have described the tools we will use to attack and optimize some of our problems. These tools allow us to maximize the performance of our computing systems. Along this section we will show three examples in which we have been working recently. Two of them are within the framework of collision physics and the last one is inside the laser–matter interaction area. Even when these three examples are inside the subject of atomic and molecular physics, we are completely confident that our *recipe* would be easily extending to other problems both in other branches of physics and other sciences. It is worth of mention that, on one hand, we have used in all cases our older and largely proved routines and programs implemented in an serial environment and, on the other hand, our approach is not restricted to a particular programming language.

3.1 Fully Differential Cross Sections calculations for single ionization of helium by ion impact

Starting a decade ago with the development of the experimental technique known as COLTRIMS (Cold Target Recoil Ion Momentum Spectroscopy), the collision physics by ion impact has attracted substantial attention. In these experiments, all the particle momenta are measured in a *single shot*. In the case of a single ionization processes the ionized electron momentum, the residual–ion one (also known as recoil–ion) and, using momentum conservation, the projectile transfer momentum, are simultaneously obtained. Consequently we have a complete description of such collision events. For this reason it is important to develop theoretical approaches that allow us to describe in detail these atomic processes. The magnitude measured in the experiments is known as Fully Differential Cross Sections (FDCS). We will not enter in greater detail of the theoretical models used to calculate the FDCS, only we will concentrate in the most important parameters at the time of evaluating numerically FDCS.

Since we are considering single ionization of helium by ion impact, the parameters that we will leave fixed in each process are: an atomic model for the helium and the projectile charge and velocity, that corresponds to a given impact energy. FDCS is function of the momentum transfer q, that can be associated directly to the projectile scattering angle and to the energy transferred to the atomic system, and of the electron variables, i.e. ionized electron energy E_e and electron ejection angles θ_k and ϕ_k. Consequently, we will use as parameters in our input files different values of q, E_e, θ_k and ϕ_k. Concerning to the angular variables, we will choose the so–called collision plane to directly compare with the experimental results (Schulz *et al.* (2003)). In this way in each of our programs we will fix the value of ϕ_k and we will vary θ_k suitably, for example calculating 30–40 points between the range of $0 \leq \theta_k \leq 2\pi$. Consequently, our *input variables* will be different values of the moment transfer q and the ionized electron energy E_e.

Following Ciappina and Cravero (2006) we see that it is necessary to perform calculations of FDCS for single ionization of helium by 2 MeV/amu C^{6+} using the parameters: $E_e = 1$ eV, $q = 0.45, 0.65, 1.0, 1.5$ a.u.; $E_e = 4$ eV, $q = 0.45, 0.65, 1.0, 1.5$ a.u. y $E_e = 10$ eV,

$q = 0.45, 0.65, 1.0, 1.5$ a.u. and for different theoretical approaches (in such example FBA and CDW–EIS). That means to execute the same routine a dozen times. For example, for the first 12 executions of our FDCS calculation program in one specific theory we have:

```
qsub -t 1-12 fdcs.sh
```

where `fdcs.sh` is our *script file* that could be exemplified as follows

```
#!/bin/bash
#$ -o $HOME/fdcs/
#$ -e $HOME/fdcs/
#$ -l h_rss=1500M,h_fsize=3000M,h_cpu=900:00:00,
s_cpu=899:59:00,hw=x86_64
$HOME/wd/fdcs.exe < $HOME/fdcs/data$SGE_TASK_ID.in
> $HOME/fdcs/fdcs$SGE_TASK_ID.out
```

Here the input files `data1.in-data12.in` would contain

`data1.in`	`data2.in`	`data3.in`	`data4.in`
1	1	1	1
0.45	0.65	1.0	1.5
`data5.in`	`data6.in`	`data7.in`	`data8.in`
4	4	4	4
0.45	0.65	1.0	1.5
`data9.in`	`data10.in`	`data11.in`	`data12.in`
10	10	10	10
0.45	0.65	1.0	1.5

On the other hand, the output files `fdcs1.out-fdcs12.out` yields, for example, the results showed in Ciappina and Cravero (2006), i.e. two columns containing the electron angle θ_k and the corresponding FDCS (in suitable units).

Let us show how is the scalability of our distribution technique. In Fig. 1 we plot the computational time elapsed in FDCS calculations as a function of the number of processors, for different theoretical approaches.

We can observe the excellent scalability of our technique. Meanwhile using a single processor the elapsed time for the 12 FDCS calculations in CDW–EIS for the first set of results of Ciappina and Cravero (2006) is about of 700 seconds, the same calculations using 12 Opteron AMD (64 bits) with 2 GB of memory each and a clock speed of 2.6 GHz, take only 60 seconds, i.e. one order of magnitude less. Even when we have used here only a dozen of input files, in principle there are not limitation in the quantity of elements in our array of *jobs*. Obviously, the elapsed total time will depend of each particular cluster and of its load balance, since if we send for execution an array with more elements than available processors, there will be waiting times that will contribute to our productivity loss.

3.2 Dalitz plots calculations using Monte Carlo techniques to assess two– and three–body interactions in single ionization of helium by ion impact

Our second example is based in Ciappina *et al.* (2006). In such work, the Dalitz plots, used intensively within nuclear and particle physics, are implemented to assess the two– and three–body interactions in single ionization of helium by ion impact. From a computational point of

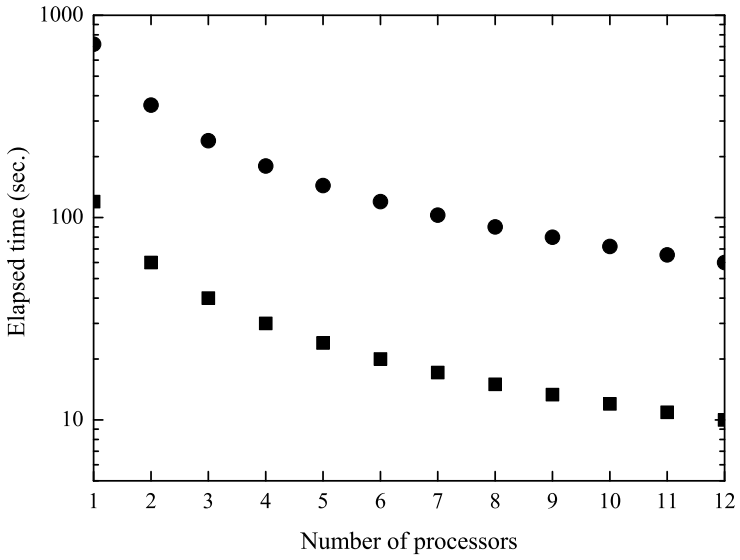

Figure 1: Elapsed time in computations of FDCS for single ionization of helium by ion impact vs. number of processors. Circles: FDCS in CDW-EIS from Ciappina and Cravero (2006), squares: FDCS in First Born Approximation (FBA).

view, this problem represents a big challenge since, in order to make these graphics, it is necessary to calculate Double Differential Cross Sections (DDCS), which are obtained integrating the FDCS explained in the previous Section, tens and hundreds millions of times to obtain a good agreement with the experimental data (Schulz *et al.* (2004)). This task resembles the usual Monte Carlo approach, in the sense that random numbers are used to model a physical process. The procedure which we have used in this example is the following. First we have performed tests for different number of events, i.e. DDCS for random values of the input parameters. These task allow us to (i) qualify our random number generators and (ii) estimate the calculation time as a function of the event number. In Fig 2. we can observe the computational time for values of 10^5, 10^6, 10^7 and . We should to say that in order to obtain a good agreement with the existing experimental data, at least 10^8 events or more are necessary. In order to obtain a Dalitz plots of this type with a CDW–EIS theory in a single processor we would have required 1 month of CPU time!. Here, our distribution scheme shows its maximum power.

Once we have chosen adequately our random number generator, the following step will be to distribute the calculation in such a way to obtain results in a reasonable time. In this case our input files `data1.in-dataN.in` will contain only the *random seeds* to *feed* the random number generators and we will divide the total number of events N_{ev} by the number of processors in which we are going to send to execute our program. For example, if we have 100 available processors, a reasonable number nowadays in medium–size clusters, we will be able to calculate a Dalitz plots with 10^8 events in 10-12 hours, depending on the availability and speed of the different nodes. This dramatic drop in the computational time has allowed us to make many tests for different theories and parameters as can be seen in Ciappina *et al.* (2006).

In a first view it could seem tedious to have to create, for example, 100 input files, but exists numerous tools, both in the Linux operating system and in different programming languages, that simplify this task. On the other hand, the management of the hundreds of output files does

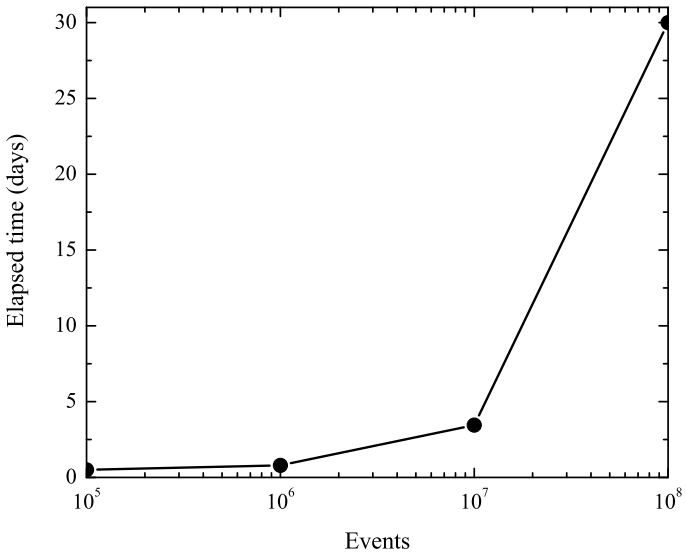

Figure 2: Elapsed time in a Dalitz plot calculation for single ionization of helium by ion impact vs. the number of events. Circles: DDCS in CDW–EIS theory (see text and Ciappina *et al.* (2006) for more details).

not represent a problem, since we only should to *add up* the results obtained for the DDCS in each node for a specific point of a given Dalitz plot.

3.3 High-order harmonic generation in complex molecules using the Strong Field Approximation

Our final example is within the context of laser–matter interaction. We show how is possible to employ our technique to a some kind of time–dependent problem: the high–order harmonic generation (HHG). Complex atoms and molecules interacting with strong ultrashort laser pulses are interesting systems to study many-body effects in an external field. Among these, the occurrence of interference effects arising from the atomic centers in a di- or polyatomic molecule is one of the most vigorously pursued research topics. Furthermore, footprints of the interference effects have been observed in other laser–induced processes, e.g in above threshold ionization (Grasbon *et al.* (2001); Jaroń-Becker *et al.* (2004)) and in high harmonic generation spectra of small molecules (Lein *et al.* (2002); Baker *et al.* (2006); Vozzi *et al.* (2005)). We present a theoretical approach of high harmonic generation in complex molecules, emphasizing the numerical aspects of the problem and we show our results for CO_2 and C_{60}.

The process of HHG, in which an atom or a molecule emits radiation at multiples of the driving laser frequency, can be understood in terms of the so-called Lewenstein or three–step model (Lewenstein *et al.* (1994)). The first step is the strong field ionization of the atom or molecule as a consequence of the nonperturbative interaction with the coherent electromagnetic radiation. The classical propagation of this electron in the field defines the second step of the model. Finally, in the third step the electron recombines under the emission of a high-energy photon. One of the main features of the HHG process is its coherence, that means the emitted radiation can be used to create coherent UV or XUV pulses. Nowadays this short-wavelength emitters promise delightful applications on the attosecond time scale.

Ab-initio numerical simulation of HHG in complex systems represents a challenge from the computational point of view. Therefore, such calculations are restricted to simple diatomic systems, such as H_2^+ and H_2. At present, the response of more complex molecules can be analyzed using approximate theories and models only. Such molecules are however of particular interest in view of recent experimental data (Baker *et al.* (2006); Vozzi *et al.* (2005)).

Applying the Lewenstein ansatz (Lewenstein *et al.* (1994)) and using the single active electron (SAE) approximation, the time–dependent dipole moment of a molecular system driven by an external field is given by:

$$\begin{aligned}\mathbf{D}(t,\mathbf{R}_1,\ldots,\mathbf{R}_M) &= -i\int_0^t \mathrm{d}t' \int \mathrm{d}^3p\, \mathbf{d}^*_{\mathrm{rec}}(\mathbf{p}+\mathbf{A}(t),\mathbf{R}_1,\ldots,\mathbf{R}_M) \\ &\quad \times d_{\mathrm{ion}}(\mathbf{p}+\mathbf{A}(t'),\mathbf{R}_1,\ldots,\mathbf{R}_M,t')\exp[-\mathrm{i}S(\mathbf{p},t',t)] + \mathrm{c.c.}, \quad (1)\end{aligned}$$

with $S = \int_{t'}^{t} \mathrm{d}t''\{[\mathbf{p}+\mathbf{A}(t'')]^2/2 + I_{\mathrm{p}}\}$ being the semiclassical action, I_{p} the ionization potential of the target system in its electronic ground state and $\mathbf{A}(t) = -\int_{-\infty}^{t}\mathbf{E}(t')\mathrm{d}t'$ is the vector potential, where $\mathbf{E}(t)$ is the electric field. The spectrum of the emitted light polarized along a certain direction $\hat{e}$ is obtained by modulus squaring the Fourier transform of the dipole acceleration along $\hat{e}$,

$$\hat{e}\cdot\mathbf{a}(\Omega) = \int_0^{T_{\mathrm{p}}} \mathrm{d}t\, \mathbf{e}\cdot\ddot{\mathbf{D}}(t)\exp(i\Omega t), \quad (2)$$

where the integration is carried out over the duration of the laser pulse T_{p}. Due to the anisotropy of the molecular system, in contrast to atoms, the emitted radiation can be polarized along other directions than the laser polarization axis. Here, we consider the harmonic radiation polarized along the direction of the laser electric field, $\hat{\mathbf{x}}$, i.e. $a_x(\Omega) = \hat{\mathbf{x}}\cdot\mathbf{a}(\Omega)$.

In Eq.(1), d_{ion} ($\mathbf{d}_{\mathrm{rec}}$) defines the ionization (recombination) amplitude (Lewenstein *et al.* (1994); Altucci *et al.* (2006)) and represents the quantal mechanical formulation of an ionization (recombination) event. The key point in these quantities is the initial molecular molecular wavefunction, which we represent as a linear combination of atomic orbitals centered at the nuclear position $\mathbf{R}_1,\ldots\mathbf{R}_M$, $\Phi(\mathbf{r},\mathbf{R}_1,\ldots,\mathbf{R}_M) = \sum_{i=1}^{M}\sum_{j=1}^{j_{max}} a_{i,j}\phi_{i,j}(\mathbf{r},\mathbf{R}_i)$, with M is the total number of the nuclei in the molecule, $a_{i,j}$ are the variational coefficients of the atomic functions ϕ_i, and j_{max} is the size of the basis set used. The orbitals are obtained using the quantum chemical self-consistent Hartree-Fock GAUSSIAN and GAMESS code. Using such codes is possible to obtain numerically atomic and molecular wavefunctions with a given degree of accuracy, adding different complexity in the particles interactions. The mathematical modeling of ground state molecular wavefunctions is a big challenge, since it is necessary to deal with many-body quantum mechanical problem. The utilization of chemical codes allow us to simplify enormously this problem and to concentrate our effort in the subsequent physical phenomena. Thus, the dipole moment in Eq. (1) takes account of the structural symmetry, the multi-center nuclear positions and the angular momentum of the molecular orbital.

In practice, the Lewenstein model in molecules provides more accurate results when the momentum expectation-value (which we refer to here as the velocity form) is used to calculate the dipole acceleration. This is exemplified by the spectra for an aligned CO_2 molecule in Fig. 3. Both the results, obtained in the velocity (panels (a) to (c)) as well as in the length (panels (d) to (f)) gauge, show characteristic minima, which appear due to an interference of the electron waves emitted from the two O atoms. Please note, that in the highest occupied molecular orbital

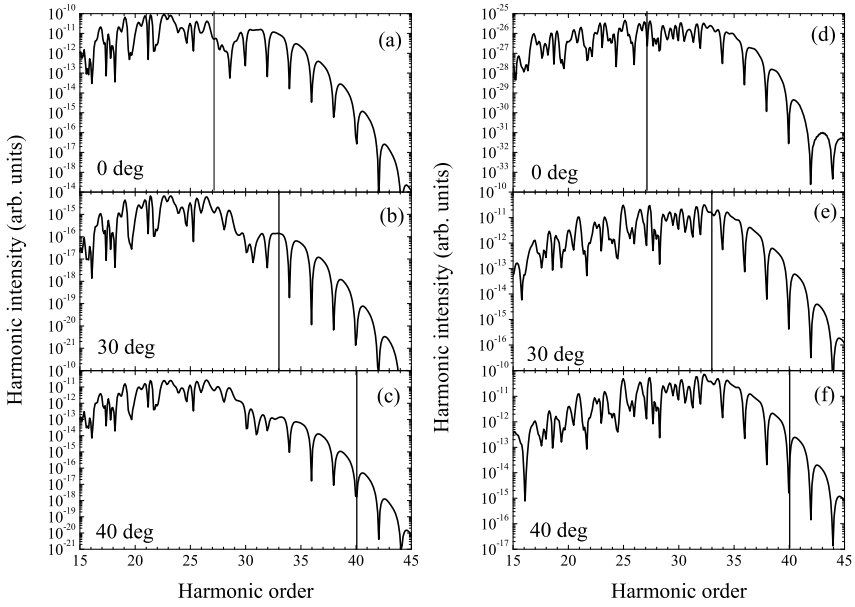

Figure 3: Harmonic spectra of a CO_2 molecule interacting with an laser pulse at a peak intensity of 2×10^{14} W/cm^2, a pulse duration of 30 fs and a wavelength of 800 nm. Panels (a), (b) and (c) show the results obtained in the velocity gauge, while those in panels (c), (d) and (e) are calculated in the length gauge. The vertical lines show the minima predicted from the two-slit interference formula (Lein *et al.* (2002)).

of CO_2 the contributions from the C atom are negligible and the molecule appears as a two-center emitter. As can be seen from the Fig. 3, for the velocity gauge the position of the minima agree well with the predictions of the two-slit interference formula (Lein *et al.* (2002)). This observation is in agreement with recent experimental results (Vozzi *et al.* (2005)). In contrast, the results obtained in length gauge fail to reproduce the two-slit interference results.

We have applied the above theory in velocity gauge to calculate HHG spectra for the fullerene C_{60} at long wavelengths (Ciappina *et al.* (2007)). First results for an ensemble of randomly oriented molecules are shown in Fig. 4. Interestingly, the spectrum exhibits pronounced minima too. They are due to a multi-slit interference effect of the partial electron waves emanating from the 60 nuclear centers. We have confirmed this interpretation in test calculations, in which we have added the contributions from the different nuclear centers incoherently and indeed the minima disappeared. We expect that the observed oscillations and multislit behavior in the HHG spectrum shows up for other fullerenes having a spherical structure with a large radius too.

From a numerical point of view the main task is the calculation of the time dependent dipole moment (Eq. (1)) and its subsequent Fourier Transform (Eq.(2)). In our example we divide the pulse duration T_p in, for example, 2^{14} =16384 points. If we equally split this interval and send for execution in different processors, we note that the first processes will finish quickly, compared with the subsequent ones, since the integration start in 0 and the final integration time will increase successively. To avoid this problem we employ the following technique. We divide the total number of point in steps, for example, of 128, i.e. in the first processor we calculate the points 1,128,256,etc. The second processor will work with 2,129,257, etc.

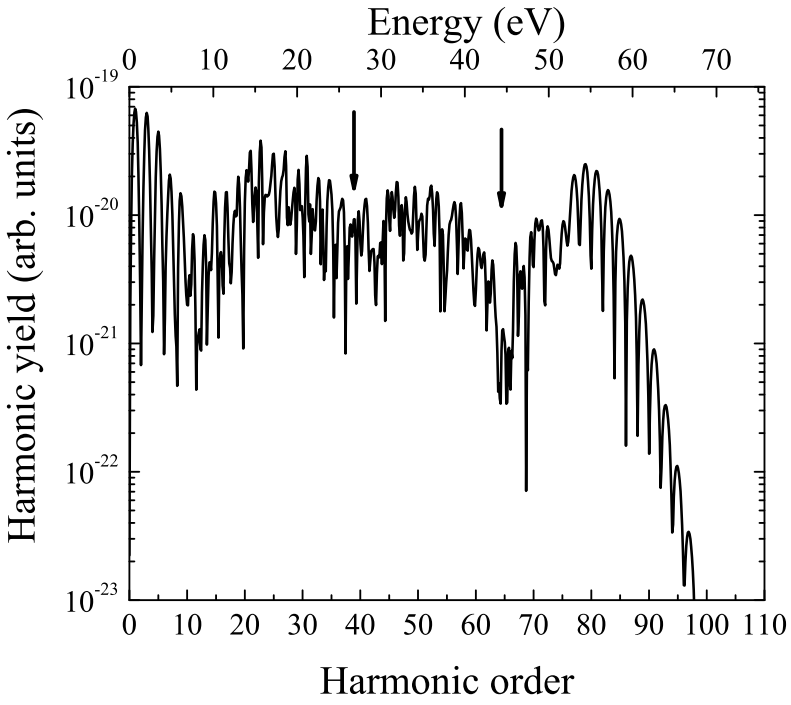

Figure 4: HHG spectrum for a randomly oriented C_{60} molecule. The laser intensity is $I = 5 \times 10^{13}$ W/cm^2, duration 50 fs and wavelength $\lambda = 1800$ nm. The arrows indicates the minima predicted using a spherical model for the ionization and recombination steps.

The whole scheme in the calculation Eq. (1) would be. First we send for execution one array of 128 elements,

```
qsub -t 1-128 hhg.sh
```

where `hhg.sh` is our *script file* that has the same characteristic than in our other cases. The input files in this problem are quite simple, since they only contain natural consecutive numbers from 1 to 128. In that way, the results of each program will be 128 points equally separated in time. Afterward, we have to collect all the output files and to perform a time ordering. The final step is the Fourier Transform calculation, that can be done directly using some of the FFT routines existing in the free–software market.

4 CONCLUSIONS AND PERSPECTIVES

We have shown throughout this work that it is possible to perform a large number of numerical calculations concurrently using our older serials routines, together with simple commands provided by the software installed to manage and optimize clusters under Linux operating system.

The intensive utilization of these tools has allowed us to solve problems impossible to deal with in other ways, as it can be seen from the examples of the last Section. Furthermore we were able to make a large number of simulations varying the relevant parameters in each specific problem. Often these possibilities are not known by habitual users of old codes, who have remained with the parallelization paradigms of the 80' or early 90's, when the main (or even only) route to parallelization implied radical modifications of the programs' codes.

The distribution technique described in this work, provided by tools available at the operating system level, would allow useful and reliable codes to enjoy the benefits of the modern hardware that clusters provide, with a negligible amount of additional programming effort.

Acknowledgments

This work is partially sponsored by PICTO UNS 931 of the ANPCyT and by the PGI Nr. 24/034 of the Universidad Nacional del Sur, Argentina. MFC acknowledges the Visitors Program of the Max Planck Institute for the Physics of Complex Systems for financial support.

REFERENCES

C. Altucci *et al.* High-order harmonic generation in alkanes. *Phys. Rev. A*, 73:043411, 2006.

S. Baker *et al.* Probing proton dynamics in molecules on an attosecond time scale. *Science*, 312:424, 2006.

M.F. Ciappina and W.R. Cravero. Fully differential cross sections for C^{6+} single ionization of helium: the role of nucleus-nucleus interaction. *J. Phys. B: At. Mol. Opt. Phys*, 39:2183–21984, 2006.

M. F. Ciappina *et al.* Multislit interference patterns in high-order harmonic generation in C_{60}. *submitted*, 2007.

M.F Ciappina *et al.* Theoretical description of two– and three–particle interactions in single ionization of helium by ion impact. *Phys. Rev. A*, 74:042702, 2006.

G. W. F. Drake. *Springer Handbook of Atomic, Molecular and Optical Physics*. Springer, 2006.

F. Grasbon *et al.* Signatures of symmetry-induced quantum-interference effects observed in above-threshold-ionization spectra of molecules. *Phys. Rev. A*, 63:041402, 2001.

A. Jaroń-Becker *et al.* Ionization of N_2, O_2, and linear carbon clusters in a strong laser pulse. *Phys. Rev. A*, 69:023410, 2004.

M. Lein *et al.* Role of the intramolecular phase in high-harmonic generation. *Phys. Rev. Lett.*, 88:183903, 2002.

M. Lewenstein *et al.* Theory of high–harmonic generation by low–frequency laser fields. *Phys. Rev. A*, 49:2117, 1994.

G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, page 8, 1965.

M. Schulz *et al.* Three–dimensional imaging of atomic four–body processes. *Nature*, 422:48, 2003.

M. Schulz *et al.* Two-particle versus three-particle interactions in single ionization of helium by ion impact. *J. Phys. B: At. Mol. Opt. Phys.*, 37:4055–4067, 2004.

Sun Microsystems Inc. *Sun^{TM}ONE Grid Engine Administration and User's Guide*. Sun Microsystems Inc., 2002.

C. Vozzi *et al.* Controlling two–center interference in molecular high harmonic generation. *Phys. Rev. Lett.*, 95:153902, 2005.