Calculation of Phonon Dispersions on the Grid Using Quantum ESPRESSO

Riccardo di Meo[1], Andrea Dal Corso[2,3], Paolo Giannozzi[3,4] and Stefano Cozzini[3*]

[1] *The Abdus Salam International Centre for Theoretical Physics, Trieste, Italy*
[2] *SISSA-ISAS, Trieste, Italy*
[3] *CNR-INFM DEMOCRITOS National Simulation Center, Trieste, Italy*
[4] *Dipartimento di Fisica, Università di Udine, Udine, Italy*

Lecture given at the
Joint EU-IndiaGrid/CompChem Grid Tutorial on
Chemical and Material Science Applications
Trieste, 15-18 September 2008

LNS0924010

[*]cozzini@democritos.it

Abstract

We describe an application of the Grid infrastructure to realistic first-principle calculations of the phonon dispersions of a relatively complex crystal structure. The phonon package in the Quantum ESPRESSO distribution is used as a computational engine. The calculation is split into many subtasks scheduled to the Grid by an interface, written in phython, that uses a client/server approach. The same interface takes care of collecting the results and *re-scheduling* the subtasks that were not successful. This approach allows the calculation of the complete phonon dispersions in materials described by a relatively large unit cell, that would require a sizable amount of CPU time on a dedicated parallel machine. Our approach decouples the application from the underlying computational platform and can be easily used on different computational infrastructures.

Contents

1 Introduction

The efficient and reliable execution of realistic scientific applications on the Grid infrastructures is still far from being trivial. Grid computing paradigm defines such infrastructures as a collection of geographically distributed computational resources glued together by a software (called "middleware") in order to allow users to access and use them in a easy and transparent ways. Different types of middleware exist, like for instance the BOINC toolkit, which animates many Grids that usually rely on volunteer computing (like Seti@Home or Protein@Home), or e.g. other Grids based on Java or proprietary solutions. The European Union funded EGEE (Enabling Grids for E-sciencE) project [1], offers nowaday one of the largest distributed computing infrastructure (around 100K cpus) for scientists. EGEE developed its own middleware that made the use of widely geographically distributed networks more accessible. However such middleware still does not provide the features required to run parallel simulations reliably, efficiently and with minimum effort from users side. Such applications require the creation of additional software to deal with as many technical details as possibile, thus leaving for the researcher free to focus on the scientific aspect of the problem.

In this paper we report our experience with realistic computations of material properties using the Quantum ESPRESSO (Q/E) [2] package. Q/E is an integrated suite of computer codes for electronic-structure calculations and materials modelling [3], implementing Density-Functional Theory in a plane-wave basis set. Typical CPU and memory requirements for Q/E vary by orders of magnitude depending on the type of system and on the calculated physical property, but in general, both CPU and memory usage quickly increase with the number of atoms in the unit cell. As a consequence, running Q/E on the Grid is not a trivial task: only tightly-coupled MPI parallelization with memory distribution across processors [4] allows to solve "large" problems, i.e. systems requiring a large number of atoms in the unit cells. The resulting MPI program requires fast communications and low latency and does not fit with the standard MPI execution environment provided by the EGEE computational infrastructure. The default MPI implementation is in fact based on MPICH over Ethernet and does not provide enough performance in terms of latency and bandwidth, especially in the case of farm-like Computing Elements.

Running Q/E on the Grid is still possible, but limited at present to calculations that fit into the memory of a single Computing Element. There are, however, some cases in which loosely-coupled parallelization available on the Grid can be really useful. Q/E currently implements, using MPI, at least one such case [4]: parallelization over "images", i.e. points in the configuration space, used for the calculation of transition paths and energy barriers. The number of images is however typically small, in the order of 10 at most. A case that looks more promising for Grid execution is the calculation of the full phonon dispersions in crystals. Medium-size systems (a few tens of atoms) will easily fit into a typical Computing Element, but many independent calculations, in the order of hundreds, will be needed, thus making the overall computational requirement quite large. In this paper we describe in some detail how to run such a calculation, the porting procedure and the results obtained so far. The paper is organized as follows: In section 2 we describe in some detail how to compute the full phonon dispersions in crystals using Q/E. Section 3 reports the computational analysis and requirements of the phonon calculations. Section 4 describes the subsequent implementation on the Grid by means of a client/server architecture. Section 5 reports a benchmarking analysis where computational efficiency attained on the Grid is compared against High Performance Computing (HPC) facilities. Finally, section 6 draws some conclusions and outlines future perspectives.

2 Phonon calculation

Phonons in crystals [5] are extended vibrational modes, propagating with a wave-vector $\mathbf{q}$. They are characterized by a vibrational frequency $\omega(\mathbf{q})$ and by the displacements of the atoms in one unit cell. The $\mathbf{q}$ wave-vector is the equivalent of the Bloch vector for the electronic states and is inside the first Brillouin zone, i.e. the unit cell of the reciprocal lattice. Phonon frequencies form "dispersion bands" quite in the same way as electronic states. For a system with N atoms in the unit cell, there are $3N$ phonons for a given $\mathbf{q}$. The dynamical matrix contains information on the vibrational properties of a crystal: phonon frequencies are the square roots of its eigenvalues while the atomic displacements are related to its eigenvectors.

Q/E calculates the dynamical matrix of a solid using Density-Functional Perturbation Theory [6]. In this approach, one calculates the charge response to lattice distorsions of definite wave-vector $\mathbf{q}$. The starting point is the electronic structure of the undistorted crystal, obtained from a conventional Density-Functional Theory self-consistent (scf) calculation. A different charge response must be calculated for each of the $3N$ independent atomic displacements, or for any equivalent combination thereof. Q/E uses atomic displacements along symmetry-dependent patterns the *irreps* (shortend for "irreducible representations"). The irreps are sets of displacement patterns that transform into themselves under *small group* of **q**, i.e. the symmetry operations that leave both **q** and the crystal unchanged. Since the irreps of the small group of **q** are typically 1- to 3-dimensional, only a few displacement patterns belong to one irrep and only the responses to these patterns need to be simultaneously calculated. This procedure allows to exploit symmetry [3] in an effective way, while keeping the calculation of the charge response within each irrep independent from the others. Once the charge response to one irrep is self-consistently calculated, the contribution of this irrep to the dynamical matrix is calculated and stored. When all atomic displacements (or all irreps) have been processed, the dynamical matrix for the given **q** is obtained.

In order to calculate the full phonon dispersions, and thus all quantities depending on integrals over the Brillouin Zone, one needs dynamical matrices for any **q**−vector. In practice, one can store the needed information in real space under the form of Interatomic Force Constants [7]. These are obtained by inverse Fourier Transform of dynamical matrices, calculated for a finite uniform mesh of **q**−vectors. The number of needed **q**−vectors is relatively small, since Interatomic Force Constants are short-ranged quantities, or can be written as the sum of a known long-ranged dipolar term, plus a short-ranged part. Once Interatomic Force Constants in real space are available, the dynamical matrix can be reconstructed at any desired value of **q** with little effort. Alternatively, one can compute a finite number of **q**−vectors and plot or interpolate the resulting phonon dispersion branches. We stress here that phonon calculations at different **q** are independent.

3 Computational analysis of the phonon calculation

Crystals with unit cells containing a few tens of atoms, up to ~ 100, fit into a single modern computing element and require relatively short execution time (minutes to hours) for the scf step (code `pw.x` of the Q/E distribution). The memory requirement of a phonon calculation is somewhat larger than that for the scf calculation, but of the same order of magnitude. Instead a full-fledged phonon calculation for a system of N atoms per unit cell for a uniform mesh with n_q $\mathbf{q}-$vectors will require a CPU time at least as large as $\sim 3Nn_q$, times the CPU time for the scf step. For systems with a few tens of atoms in the unit cell, this multiplicative factor can be in the order of thousands and more. As a consequence, phonon calculations for crystals of more than a few atoms are considered HPC applications.

In Q/E the n_q dynamical matrices for a uniform mesh of $\mathbf{q}$ vectors are calculated by the `ph.x` code. The `ph.x` code can split the phonon calculation on the mesh of n_q points into $\tilde{n}_q$ runs, one for each of the $\tilde{n}_q$ symmetry- inequivalent $\mathbf{q}$ vector of the mesh. In order to better distribute the computation, we have modified the `ph.x` code to calculate separately also the contribution of each irrep to the dynamical matrix and to save it into a file. Assuming that there are $N_{irr}(\mathbf{q}_i)$ irreps for $\mathbf{q} = \mathbf{q}_i$ the complete phonon calculation can be split into up to $N_{tot} = \sum_{i=1}^{\tilde{n}_q} N_{irr}(\mathbf{q}_i)$ separate calculations, that can be simultaneously executed into different Computing Elements. The CPU time required by each response calculation, i.e. for each irrep at one $\mathbf{q}-$vector, is roughly proportional to (and typically larger than) the dimension of the irrep, times the CPU time required by the starting scf calculation for the undistorted system, times the ratio $N_G/N_\mathbf{q}$ between the dimension number of symmetry operations of the point group N_G of the crystal and the dimension $N_\mathbf{q}$ of the small group of $\mathbf{q}$. The latter factor is of no importance in low symmetry crystals for which $N_G = N_\mathbf{q} = 1$ but it can be quite large (up to 48) for a general $\mathbf{q}$ in a highly symmetric solid. In the latter case, once the dynamical matrix at $\mathbf{q}$ has been calculated, we can calculate without effort also the dynamical matrices of the star of $\mathbf{q}$ that contains $N_G/N_\mathbf{q}$ points so the total amount of time for computing the n_q points of the mesh is independent upon the symmetry. However, depending on the system, the efficiency of the Grid partitioning can vary from cases in which each `ph.x` run requires approximately the same CPU time as the scf calculation, to particularly unfortunate cases in which a `ph.x` run will require up to ~ 50 times the CPU time of a single scf run.

The contribution of each irrep to the dynamical matrix – typically a few Kbytes of data – is written to a `.xml` file which can easily be transferred among different machines. After collecting all the `.xml` files in a single machine, a final run of `ph.x` can collect the dynamical matrices in the n_q $\mathbf{q}$ points. This has been implemented with the following approach. The files with the contribution of a given irrep to the dynamical matrix are written in the directory `outdir/prefix.phsave` and are called `data-file.xml.q.irrep` where `q` identifies the $\mathbf{q}$ point and `irrep` the irrep. When the `ph.x` code is started by setting the input variable `recover` to `.true.` it will check in the directory `outdir/prefix.phsave` for the existence of files called `data-file.xml.q.irrep` and when a file is found the contribution to the dynamical matrix of the corresponding irrep is not recalculated but read from a file. When the files `data-file.xml.q.irrep` for all `q` and all `irrep` are present in the directory, `ph.x` will only collect the dynamical matrices and diagonalize them.

Four new input variables control the phonon run: `start_q` and `last_q` choose among the $\tilde{n}_q$ points those actually calculated in the current run. `start_irr` and `last_irr` choose the irreps. In the present version, the `ph.x` code evaluates the displacement patterns of each irrep by an algorithm based on random numbers. In order to guarantee the consistency of irreps across different machines we calculate the displacement patterns in a preparatory run and write them to an `.xml` file that is then sent to all the computing elements. The file `data-file.xml.q` contains the displacement patterns for all irreps of a given **q** and the file `data-file.xml` contains a few information to control the phonon run, the most important being the cartesian coordinates of the $\tilde{n}_q$ **q** points. If these files are found in the `outdir/prefix.phsave` directory the mesh of **q** points and the displacement patterns are not recalculated by `ph.x`.

4 Grid enabling procedure

A typical phonon calculation can be performed in three different ways: (i) in a single serial run, (ii) in $\tilde{n}_q$ independent calculations, and (iii) in N_{tot} calculations independently. Figure 1 shows the three different approaches.

For all the approaches a preparatory calculation is required: a self consistent calculation performed by Q/E `pw.x` tool finds the electronic structure of the undistorted system. The quantities calculated by this run are saved in `outdir/prefix.save`.

In the first approach `ph.x` performs all the N_{tot} calculations in sequence, one after the other in a single (serial) execution. All files `data-file.xml.q.irrep` are written in the same directory and a single instance of `ph.x` has to be called; the run could be split into several chunks in order to be executed on HPC resources with limited wall-time assigned by the queue systems but this recover procedure, which was already implemented in `ph.x`, will not concern us here.

In the second approach the $\tilde{n}_q$ points are distributed using the variables

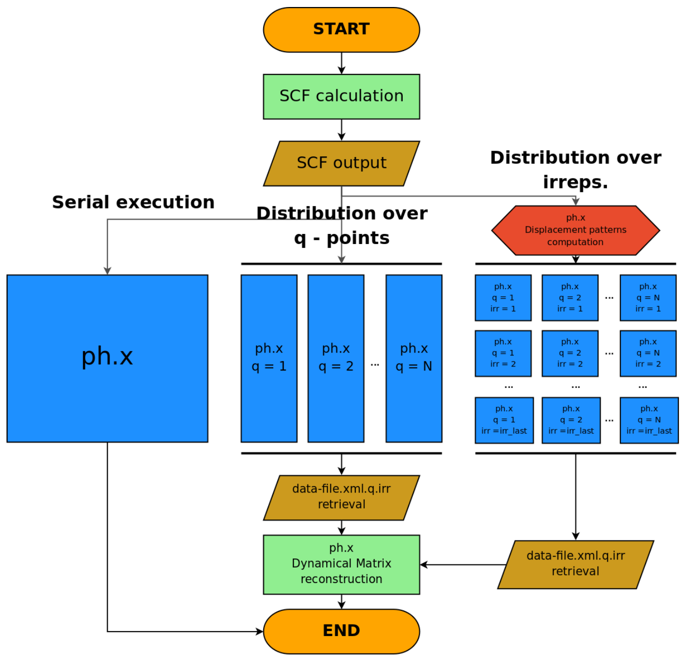

START
SCF calculation
SCF output
Serial execution
Distribution over q - points
Distribution over irreps.
ph.x Displacement patterns computation
ph.x
ph.x q = 1
ph.x q = 2
ph.x q = N
ph.x q = 1 irr = 1
ph.x q = 2 irr = 1
ph.x q = N irr = 1
ph.x q = 1 irr = 2
ph.x q = 2 irr = 2
ph.x q = N irr = 2
ph.x q = 1 irr = irr_last
ph.x q = 2 irr = irr_last
ph.x q = N irr = irr_last
data-file.xml.q.irr retrieval
ph.x Dynamical Matrix reconstruction
data-file.xml.q.irr retrieval
END

Figure 1: Flowchart depicting the different approaches to the calculation of the phonon dispersions.

`start_q` and `last_q`. This means that $\tilde{n}_q$ independent jobs are sent, each one calculating its own dynamical matrix. At this point a plot of the phonon at arbitrary **q** points can be done using specific Q/E tools `q2r.x` and `matdyn.x`.

In the third approach an additional preparatory run of `ph.x`, done by setting `start_irr=0` and `last_irr=0`, is required in order to calculate the displacement patterns of all irreps at each **q** and to write the files `data-file.xml.q` in the `outdir/prefix.phsave` directory. The `outdir` directory is then saved to a storage location accessible by means of some protocol from the available computational resources. Up to N_{tot} `ph.x` calculations can now be started simultaneously each with different values of the `start_q`, `last_q`, `start_irr` and `last_irr` input variables. The output of each `ph.x` run is one or more files `data-file.xml.q.irrep`. When all these files become available they can be collected into one `outdir/prefix.phsave` directory. A final run of `ph.x`, which does not set the variables `start_q`, `last_q`, `start_irr` or `last_irr`, collects the results and diagonalizes the dynamical matrices. Note that, in this last step, the `ph.x` code will try to recalculate the irrep if the corresponding `data-file.xml.q.irrep` file is not found in the `outdir/prefix.phsave` directory.

With respect to the sequential run, the two distributed approaches described above require additional steps to collect and recompose the dynamical matrices: the management of input and output in different directories should be done in an automatic way in order to avoid errors in the execution. In the third approach, also a final step has to be done at the end of the independent calculations to collect all the dynamical matrices. Its computational weight is negligible but it should be considered in the workflow.

The Grid porting consists in a tool that manages all the procedures sketched above as much as possible in an automatic way. This is implemented by means of a client-server architecture: a server will be contacted by different clients and will assign them slices of work. The architecture is designed so as to be portable on different computational infrastructures. Our goal is to make the tool easily usable on heterogeneous and geographically distributed resources which users can have at disposal. For this reason we decoupled as much as possible the management software from the procedures needed to obtain computational resources. We successfully tested this approach on at least two computational infrastructures: a local cluster accessible through a batch queue system and a gLite/EGEE Grid infrastructure.

Server and client software is written in python. They communicate by means of the XMLRPC protocol: the server in case of a Grid infrastructure is executed on a resolved host, usually the User Interface (UI: the machine where Grid users submit their jobs to the Grid. It is always provided with both inbound and outbound connectivity). Clients are then executed on the computational nodes, the so-called Worker Nodes (WNs). The user launches the server specifying a set of parameters (location of the binaries and data files on the Grid as well all the input needed) and then submits jobs to the computational infrastructure he wants to use. The only requirement is that the computational nodes should be able to contact back the server: this outbound connectivity request is always satisfied in case of EGEE/gLite Grid infrastructure.

As soon as a job lands on a Worker Node, the client starts executing: it first contacts back the server and requests the location of the needed data and the executable. Different protocols are supported in order to be able to store the data on different kinds of storage. In case of Grid computing `gsiftp` and `lfc` are both supported so data can be stored on Grid storage facilities like Storage Element and LFC Catalog.

Once data and executable are available, the client asks for a chunk of work to be done. This will require to assign suitable values to the four variables `start_q`, `last_q`, `start_irr` and `last_irr`. Once the calculation is performed, the output (i.e. the files `data-file.xml.q.irrep` produced in the run and possibly the output of `ph.x`) is sent back to the server which assigns another task. Client activity is terminated when it hits the wall-time limit imposed by the infrastructure (or if no active server is found). The server will stop when all the data needed to compute the required dynamical matrices have been sent back by the clients. At that point, the user will find in the server's directory `data-file.xml.q.irrep` files for all $\tilde{n}_q$ $\mathbf{q}$ points and for all irreps. The dynamical matrices can then be recomposed locally with a final call to `ph.x`.

In order to start the execution of our system the user has to provide:

- The `ph.x` binary, compiled to suit the computational infrastructure, located in a storage location reachable by the WNs.
- The output data files of the self-consistent computation from `pw.x` and of the preparatory `ph.x` run, again located in a storage reachable by the WNs.
- A template of the input file of `ph.x` which will be used by the server to create "ad hoc" input files for each client requesting tasks to be executed.

The server is then launched on the UI, where it will run until the end of the simulation, with the appropriate parameters. It is left to the user to submit an appropriate number of jobs in order to recruit computational resources where to distribute the computations. Such a task is a trivial one: users just have to submit many times the same job and this task can be easily automatized.

The server keeps track of the status of the clients connected to it: if a client fails for any reason, the server will mark the task assigned to such client as "available" again and assign it to another client as soon as it becomes available.

The fault tolerant mechanism is very important in Grid infrastructure where no strict control is possible on computational nodes and job failure rate is still high. We observed a 60% percent failure rate out of 2140 clients that successfully connected to the server (as discussed in the next section).

We finally note that the above procedure runs the `ph.x` code serially on a large number of CPUs: each client (associated with a single job) lands on a different machine. Being SMP/Multicore architecture now very common as Grid computational resources it could happen very often that such serial jobs share the resources with other tasks submitted by other users. This has for sure some drawbacks in terms of performance and efficiency. We will discuss in the conclusions possible alternative solutions to this problem.

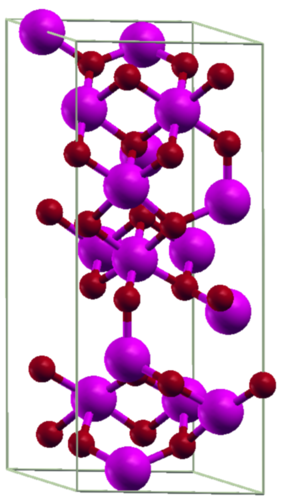
5 Results

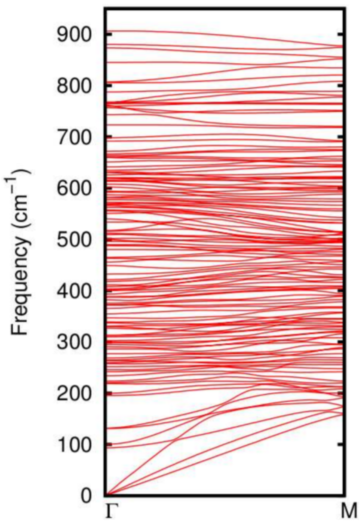
The porting procedure and the client/server architecture was developed using a few small examples. In this section we will discuss a relatively large phonon calculation which allowed us to evaluate the efficiency of the novel distributed approach against the standard HPC one.

A realistic phonon calculation on the Grid is given by the $\gamma-Al_2O_3$ system depicted here below (Fig. 2). This solid can be described as a distorted hexagonal lattice, characterized by primitive lattice vectors **a**, **b**, **c**, whose length is 5.579, 5.643, 13.67 Angstrom respectively, at angles $ab = 120$, $ac = 90$ and $bc = 89.5$. The unit cell contains 8 formula units, i.e. 40 atoms. The size of the phonon calculation is $3N = 120 \times \tilde{n}_q$ linear-response calculations. In this example we choose to sample only a line in the reciprocal space from Γ ($\mathbf{q} = 1$) to the zone boundary along the reciprocal lattice vector $\frac{\mathbf{b}\times\mathbf{c}}{\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})}$ using a $21 \times 1 \times 1$ mesh so that $\tilde{n}_q = 11$. The point group contains only the identity so every **q** vector requires the same amount of CPU time, with the exception of the Γ point for which we calculate also the response to the electric field in order to evaluate the dielectric constant and the Born effective charges. The total time needed to complete such a calculation on a single modern workstation is a few weeks. In Fig. 2b we show the output of the calculation performed on both HPC and Grid infrastructure.

We performed here the same calculation using the two different distributed approaches (over **q** and over irreps) on the HPC and Grid resources available.

Table 1 summarizes the full list of computational experiments performed with some details:


(a) Atomic structure



(b) Phonon dispersion curve

Figure 2: The $\gamma-Al_2O_3$ system.

5.1 Grid results

Let us now first discuss the performance figures obtained using the client/server mechanism presented above on the Grid (provided by the EGEE project through the `CompChem` Virtual Organization [8]) where the distribution was done over irreps.

The experiment was repeated three times varying the number of irreps assigned as a task to each client: this number was respectively 1, 4 and 6 for `grid1`, `grid2` and `grid3`. The three experiments were performed one after

Table 1: List of phonon calculations on HPC and Grid : q/irreps distribution.

code	kind	cpus	computational nodes	distributed
grid1	GRID	1	heterogeneous	over irreps
grid2	GRID	1	heterogeneous	over irreps, in group of 4
grid3	GRID	1	heterogeneous	over irreps, in group of 6
4cpu	HPC	4	opteron dual-core 2.4Ghz	over q
8cpu	HPC	8	2 x opteron dual-core 2.4Ghz	over q
16cpu	HPC	16	4 x opteron dual-core 2.4Ghz	over q

the other without overlaps, in order to maximize the resources available for each one.

The server was activated on the User Interface and about 3000 independent jobs where submitted to the Grid scheduler in a few shots of 500 jobs each separated by 12/24 hours. This was done in order to avoid heavy load on the Workload Management System (WMS)[1] and to allow a better overall scheduling policy. As reported in the previous section a high failure rate of 60% of jobs that contacted the server was reported[2]. We discovered that between 30% and 40% of them failed to download the data required for the simulation while most of the others disappeared without signaling back to the server (which happens when the client hangs, or gets killed by the queue system). We are currently searching for the reasons behind such a large number of failures.

Figure 3 gives information about the evolution of the simulation. It reports the number of active processors and the number of irreps completed as function of the duration of the experiments.

It is worth to note the amount of resources collected during the three experiments: all of them were able to recruit up to more than 130 active clients at the same time[3]; since the access to the Grid is granted "on best effort basis" we consider this an excellent result, which is hardly possible on small and medium HPC clusters without privileged policies.

Concerning the overall performance we can see that `grid3` completed the simulation in 58 hours where `grid2` and `grid1` took 67 hours and 127 hours, respectively. `grid3` was also way more efficient if we consider the total numbers of hours per processor requested to complete the experiment (see figure 3c): it used slightly less resources than `grid2` and about half the resources used by `grid1` (4064 hours/processors for `grid3` versus 7100 hours/processors for `grid1`).

`grid1` overall performance was affected by a hardware failure of the server that blocked the execution for more than 20 hours. This means that all clients connected at the time of the failure to the server were lost. Once the server was restarted and jobs resubmitted simulation resumed but a lot of calculations were actually lost. It is however evident that performances of the latter two simulations outperform `grid1`. Reasons of this difference can be understood looking at detailed information about the time required to

[1]The service handling the scheduling and queuing of jobs on the EGEE-type Grids.

[2]A failure is counted each time a client exits before being able to complete any task.

[3]We consider a client active when it is able to complete at least one irrep.

Table 2: Initialization time and scf time for Grid runs averaged for $q = 1$ and remaining points.

(a) grid1

$\mathbf{q}$	init. (h)	phqscf (h)
1	859.99	157.54
2-11	252.51	174.89

CPU time (days)	141	79
% of the time	64	36

(b) grid2

$\mathbf{q}$	init. (h)	phqscf (h)
1	249.47	182.36
2-11	70.60	192.40

CPU time (days)	40	88
% of the time	31	69

(c) grid3

$\mathbf{q}$	init. (h)	phqscf (h)
1	143.37	157.73
2-11	46.48	189.55

CPU time (days)	25	86
% of the time	23	77

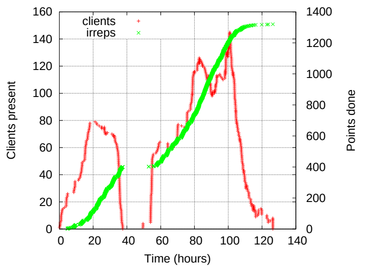
complete the irreps as reported in table 2.

We report the time and the number of jobs required to compute the full 120 irreps. We distinguish the $\mathbf{q} = 0$ vector from all the others due to the different initialization phase as already mentioned above. For the other $\mathbf{q}$ we present here the averages over the number of $\mathbf{q}$.

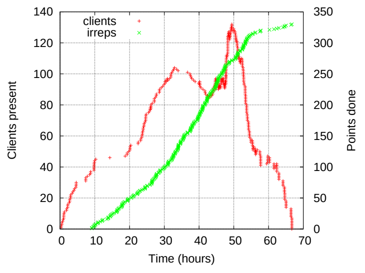
Intervals are measured by the internal timing mechanism of the `ph.x` program. The initialization time reported here is given by the `init_phq` plus `e_solve` routine just for $\mathbf{q} = 0$ vector. The linear response calculation is then done by the `phqscf` routine which accounts for all the self-consistent iterations for each irreps.

It is evident from figure 3a that in the client/server approach a considerable amount of time is spent in initialization procedures. This step for each $\mathbf{q}$ is performed, respectively, 120 times, 30 times and 20 times for `grid1`, `grid2` and `grid3`. This slows down the simulations considerably: the initialization phase alone has a major part in the CPU spent for `grid1` (141 days, accounting for 64% of the walltime). Grouping the irreps assigned to each client the problem alleviates but still has a prominent role for `grid2` and `grid3` (contributing for, respectively, 40 and 25 days, or 31% and 23% of the total time).

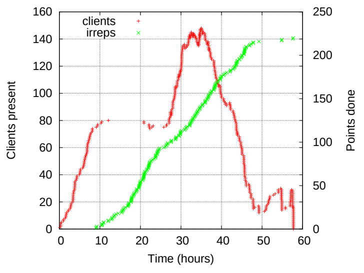
Grouping the irreps is not however the only possible way of reducing



(a) grid1



(b) grid2



(c) grid3

Figure 3: Number of irreducible representations computed and clients present over time for the Grid simulations.

the gap between our Grid oriented approach and the HPC one and early studies seem to suggest that the initialization time when distributing over the irreps might be further reduced, which would greatly contribute to the performances.

We finally note that the overhead caused by data transfer from storage elements and worker nodes (which is non-existent in a HPC environment) and vice versa is almost negligible in this context and does not affect at all the overall performances, despite the large amount of data downloaded by each client. This result is due to the fact that Grid storage allows multiple copies of the same data (“replica”) managed by a catalog (LFC central service). Such replica can be stored close to the computing elements used by the clients in the course of the simulations. This means that in many cases the data are just downloaded by a storage element on the same local area network.

5.2 HPC results

In the case of the **q** distribution approach performed on HPC resources we just submitted $\tilde{n}_{\mathbf{q}} = 11$ independent calculations through the resource manager (queue system of our HPC cluster). The code was compiled enabling MPI communications and we repeated the calculation using different numbers of cpus: 4 (a full node dual core opteron node), 8 cpus and 16 cpus (respectively two and four nodes connected via infiniband network).

Table 3: CPU time spent during on the initialization and scf time for the simulations distributed over q.

(a) 4cpu

q	jobs	init. (h)	phqscf (h)
1	8	8.16	62.02
2-11	6	5.19	60.06

CPU time (days)	10	110
% of the time	8	92

(b) 8cpu

q	jobs	init. (h)	phqscf (h)
1	4	2.41	32.13
2-11	4	1.36	30.28

CPU time (days)	5	109
% of the time	4	96

(c) 16cpu

q	jobs	init. (h)	phqscf (h)
1	2	0.81	16.01
2-11	2	0.28	15.39

CPU time (days)	2	113
% of the time	2	98

Once the jobs terminated we resubmitted them until all 120 irreps were calculated. As for the Grid, table 3 gives a detailed report of the total amount of time needed to complete 4, 8 and 16 cpus experiments. Data are presented as in table 2 with an additional column where we report the number of jobs needed to complete the full 120 irreps.

We note that at any needed restart (i.e. a new job) the initialization phase is to be repeated: therefore splitting the calculation over 10 hours chunks causes some overhead but considerably less if compared with the Grid approach.

Tables 3a, 3b and 3c gives us indications about the overall scalability of the `ph.x` code, which is acceptable: there is moreover a super-linear speed-up in the initialization phase of the calculation when increasing the number of cpus; this is just because less jobs are needed and therefore the initialization phase is computed less times.

The performances of the code when moving from 4 to 16 CPUs is also very good with an overhead of only a few percent, which implies almost perfect scalability in our range of testing.

Table 4: Walltime estimate for **q** distribution on HPC platform.

# cpu	4	8	16
total # of jobs	88	39	22
max # of concurrent jobs	10	10	8
# of submissions	9	4	3
average time waiting (m)	22	46	136
max.running time avail.	12h		
estimated time	110h	50h	40h

From the times above reported we can estimate the average duration of a complete computational experiment, taking into account the time spent waiting on a queue (based on the scheduling system information) and user policies implemented. Since the maximum wall-time allowed in our queues is 12 hours and the maximum number of jobs running is 10 (with an additional constraint that no more than 128 cpus can be used at once), we estimated the results reported in table 4.

These numbers make clear that it is far more convenient to run on 8 or 16 cpus with respect to 4. An unexpected side effect of the scheduling policy is that the time required to complete the simulation on 16cpu is only slightly smaller than on 8cpu, since the superlinear speed-up observed in the initialization phase in this case counterbalances the time spent waiting.

5.3 Comparison

We can now compare the computational efficiency of the two distributed approaches so far presented. We base this comparison on two important parameters: the total wall time required to complete the simulation that can give us the actual duration of experiment, i.e. time to result. We stress here that the duration of the experiment depends upon many different parameters, not all easy to keep under control especially in a distributed Grid environment. It therefore turns out that such value can be quite volatile and only a rough estimate can be given. We couple therefore such value with the total amount of global cputime needed (hours per processor) in order to have a more defined estimate; we also note that it is the value that supercomputer centers use to charge the use of computational infrastructures.

Concerning the duration of the experiment, we observe that a Grid infrastructure can easily compete with the HPC infrastructure for such kind of computational experiment: for instance, the HPC `8cpu` simulation is estimated to complete just 8 hours before `grid3`.

Looking at the global cputime needed we see that `grid3` is even better than HPC resources: the global amount of computing time is even slightly less than `hpc8` and `hpc16`.

A complete comparison of all the experiment performed is collected in Fig. 3.

We note here that while **q**-distributed simulations took about 115 days of CPU, `grid1` took about 220 days[4].

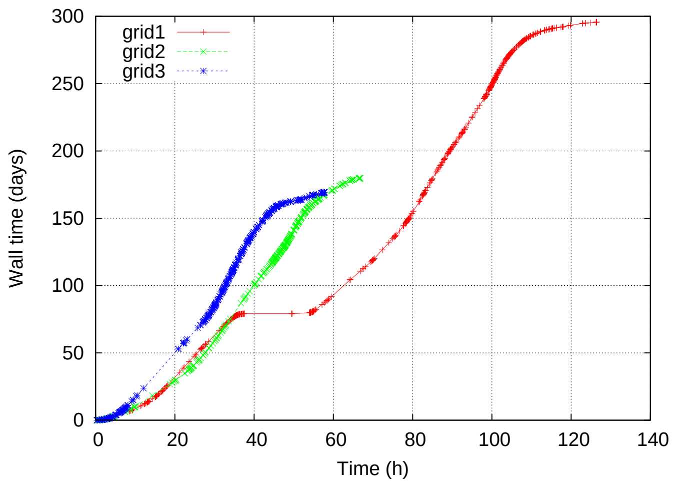


Figure 4: Resources spent on the Grid.

The reason is to be sought in the overhead caused by the initialization

[4]Without counting the CPU time spent by extra clients working on the same task at once, which accounts for the discrepancy between table 2 and figure 3.

when splitting the computation over the irreducible representations, which is also the reason behind the performance differences between the various Grid experiments.

6 Conclusions

As discussed in the previous section our approach to distribute phonon computation proved to be an interesting approach when compared with the standard HPC one. Our solution, decoupling as much as possible the scientific workflow from the recruitment of computational resources, is easily portable and perfectly interoperable with different computational platforms.

In this way, relieving the simulation from the requirement of a very fast and reliable network connection between nodes we have been able to use decoupled and un-specialized resources, like different Worker Nodes connected through the internet, to perform a task that before could be tackled only with very performing infrastructures.

The detailed performance comparison performed in the previous section highlights that times needed to complete a realistic phonon calculation on Grid infrastructure are comparable with times needed on HPC platform. We stress here the fact that computational resources are made available with a completely different policy in a Grid infrastructure with respect to an HPC environment: from this perspective our results are even more interesting and promising.

There is, however, room for making the procedure even more efficient, with a twofold approach. On one side, we can reduce the required initialization time by directly modifying the scientific code. On the other side, we can also pack more irreps together, to better fit the available queues.

We finally note that, being the multicore/SMP architecture widespread as computational resources in Grid infrastructure it would be desirable to be able to run each process on SMP node, enabling thus parallel computation of `ph.x` using the shared memory approach (the same way we performed the client/server experiment on HPC platform).

We therefore integrated in our setup another tool, developed by ICTP: the "reserve SMP nodes", which allows, through a mechanism known as "job reservation" to run codes parallelized through shared memory on the Grid, thus getting a speedup in the execution and, more importantly, allowing for larger simulations to be tackled (since a larger share of the memory on the destination machine can be reserved for the execution).

Employing such utility we have been able to run the `ph.x` executable in parallel on the Grid nodes with MPI over shmem, for instance, using two cores per node, effectively doubling the available memory for the application (as well as the computing resources), however the results obtained with other scientific applications suggest that a larger number of CPUs can be easily recruited as well. Research about the conjuncted use of reserve_smp_nodes and `ph.x` is still ongoing.

Acknowledgments. We thank Eduardo Ariel Menendez Proupin for suggesting the physical problem considered in this paper.

References

[1] Enabling Grids for E-sciencE, http://www.eu-egee.org

[2] Quantum Espresso Home Page, http://www.quantum-espresso.org

[3] P. Giannozzi *et al.*, http://arxiv.org/abs/0906.2569

[4] P. Giannozzi and C. Cavazzoni, (2009). Nuovo Cimento C 32. In press

[5] See for instance: C. Kittel, Introduction to Solid State Theory, 8th edition, Wiley, New York (2005)

[6] P. Giannozzi, S. de Gironcoli, P. Pavone and S. Baroni, (1991). Phys. Rev. B **43**, 7231

[7] S. Baroni, S. de Gironcoli, A. Dal Corso and P. Giannozzi, (2001). Rev. Mod. Phys. **73**, 515

[8] CompChem Virtual Organization, http://compchem.unipg.it/start.php