Theor Chem Acc (2007) 117:1145–1152
DOI 10.1007/s00214-006-0227-9

REGULAR ARTICLE

Gaussian grid: a computational chemistry experiment over a web service-oriented grid

N. Sanna · T. Castrignano · P. D'Onorio De Meo · D. Carrabino · A. Grandi · G. Morelli · P. Caruso · V. Barone

Received: 30 May 2006 / Accepted: 13 October 2006 / Published online: 15 December 2006

Abstract This paper describes the implementation of a quantum chemistry code on a service-oriented grid where the computational workflow is spawned over multiple, geographically distributed, sites. The application porting over the grid and its extension as a web service over local and wide area networks is fully outlined. A description of the procedures developed for this experiment is given in full detail and the applicability of the methodology to similar codes in this scientific area is assessed. The new developed, grid-aware, application has been tested by performing a comprehensive set of benchmarks including quantum mechanical calculations on the water molecule. The obtained results of the benchmarks are reported and a full comparison with respect to the parallel execution of the same tests using the standard code is discussed in detail.

Keywords Computational chemistry · Quantum chemistry · Grid computing · OGSA · Web services

1 Introduction

Computational quantum chemistry is traditionally a field which requires huge computing resources in order to solve quantum mechanical (QM) Hamiltonians describing the properties of molecular species by means of variational and/or perturbative many-body approaches. With the last decade of increasing computational applications of methods rooted into the Density Functional Theory (DFT) [1,2] and its time-dependent extension (TD-DFT) [3] to the solution of physical problems, an open debate is going on about the use of this, against the traditional *wave-function*-based quantum mechanical models. However, both families of methods remain extremely costly in terms of computing resources for the large-dimension systems of current biological and/or technological interest. The situation is even more involved for studies on condensed phases despite the development of powerful continuum [4] and discrete/continuum [5] models and the complexity further increases when going to true dynamical phenomena [6]. As a consequence, several studies are in progress with the aim of setting up innovative computational strategies able to deal in an effective way with the increasing complexity of the molecular systems of current interest, which are still domain of the classical particle simulations [7]. However, in the foreseeable future software developments will not change the general situation that leading calculations using quantum chemistry methods require a huge computing power either in terms is of CPU and/or I/O resources and top-level computer architectures are often used. Those computing machines are often built on top of commodity components and this trend seems well consolidated even for the upcoming parallel architectures. From the application side, it is now becoming evident from the analysis of the computational production in this field, that a further extension of *the local architecture concept* for High Performance Computing (HPC) is needed, and new

N. Sanna (✉) · T. Castrignano · P. D'Onorio De Meo · D. Carrabino · A. Grandi
CASPUR, Consortium for Supercomputing in Research, Via dei Tizii 6, 00185 Roma, Italy
e-mail: sanna@caspur.it

G. Morelli · P. Caruso · V. Barone (✉)
Dipartimento di Chimica and INSTM, Universitá di Napoli "Federico II", Complesso Universitario di Monte S. Angelo, Via Cintia, 80126 Napoli, Italy
e-mail: baronev@unina.it

challenging solutions should be adopted to tackle upcoming problems like the explosive growth of the biological data, stimulated by the genome projects. In this area the need of HPC solutions is growing, though usually not affordable by computational resources based on a single, even large, local parallel architecture. Grid computing addresses this problem by coordinating and unifying several computational resources [8], allowing the evaluation (and mining) of large amount of data in the teraflops (terabyte) range. Unfortunately, present-day versions of grid middleware provide only a small part of the functionality required to fully implement architecture-independent applications. On the other hand, web services are the distributed computing technology that offers powerful capabilities for scalable interoperation of heterogeneous software across a wide variety of networked platforms [9]. They give the opportunity to create a framework in which applications distributed across local and wide area network can interoperate through a set of standard protocols. The crucial difference with the past is that most of the systems consisted of ad hoc solutions (e.g. CGI programs) whereas the Web Services (WS) should lower the barrier to application integration. To increase individual and collective scientific productivity by making powerful information tools available to everyone, a service-oriented strategy is necessary. New projects on service-oriented grids [10] have the assets of both grid and web service technology and help researchers to obtain high performance web services. Complex applications exchanging huge amount of data, using several web services, have to be managed to gain high performance and high availability systems, encouraging convergence of grid and web services. Among those classes of applications, to face the problem of identifying and assessing the coding or non-coding nature of Conserved Sequence Tags (CSTs) through cross-species genome comparisons [11–13], some of the present authors recently published a grid–web service framework, CSTgrid [14], whose core is implemented as web services. The CSTgrid has been developed over an Open Grid Service Architecture (OGSA), in which services act as building blocks of the Grid system, allowing the user community to access all services without any knowledge of the underlying infrastructure. Providing high performance and high availability the CSTgrid framework can fairly handle hundreds of concurrent requests but as an open grid infrastructure based on standard grid toolkit (e.g. Globus toolkit [15]) it paves the route to new application porting. To this end, in this paper we used the underlying CSTgrid infrastructure to migrate the Gaussian package [16] execution over a geographically distributed set of nodes by encapsulating the necessary code sections into a new web service. This recently developed WS, has several new appealing features with respect to traditional HPC QM codes, i.e. among others, (1) it is architecture independent, (2) it is modular, that is, configurable for user or site requirements, (3) it is dynamical, that is, executable on available nodes. In the following, in Sect. 2 we report the essential details of the grid infrastructure used in our computational experiment, while in Sect. 3 we describe the coding of the grid software and its interface with the Gaussian parallel execution; Sect. 4 is devoted to a detailed analysis of the performance tests carried out on a distributed shared-memory cluster, and finally Sect. 5 contains the conclusions of this work.

2 The reference grid infrastructure

In this section we briefly outline the reference infrastructure of the Gaussian grid, which is based on the computing framework recently developed for the CSTgrid application.

The system is developed in multi-layered components to allow a Rapid Application Development (RAD) infrastructure and minimal administration efforts. CSTgrid is logically composed by three tiers (Fig. 1): (1)

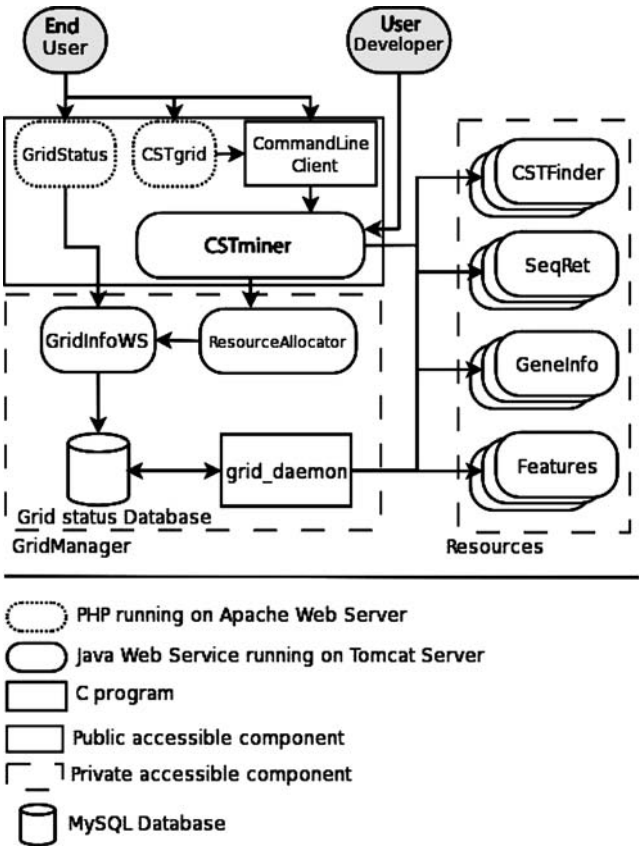

Fig. 1 The CST grid framework of reference of the Gaussian grid

an *interface tier* responsible for communicating with end-user agents such as web browsers and command-line clients; (2) a generic (not oriented to search CSTs) *grid tier* composed by a grid daemon responsible for the management of the grid resources and (3) a *resource tier* composed of a set of Resources WS, specific to search CSTs.

The interface tier is responsible for communicating with end-user agents such as web browsers and command-line clients. PHP scripts (GridStatus and CSTgrid), running under Apache, allow the user both to obtain information about the status of the grid and to launch an application job through a command-line client. Originally developed for the CSTminer application, this layer of the Gaussian grid is a public WS available to end-user developers through the standard service description layer, the Web Service Description Language (WSDL), an XML grammar for specifying a public interface for a web service.

The grid manager tier is based on four components: two web services (*GridInfo* and *ResourceAllocator*), one database to store information about the grid status and one grid daemon. The database contains all the information about the hosts taking part in the grid, the services available on that hosts and the history of the availability of these services. The history data are managed by the grid daemon, a C program running in background, which periodically queries their services to know the actual status and stores this information into the database. The detecting time interval for a given WS is calculated by the system and thus configured and stored in the database. *GridInfo* is a private web service responsible for giving access to information about the grid status toward the external world via web while *ResourceAllocator* is the web service responsible for taking resource requests and providing access to them according to a "load-balancing" failure-safe policy. It takes up-to-date information about the grid by the *GridInfo* web service. Traditional control algorithms for load balancing include Random, Round-Robin (RR) [17], Weighted Round-Robin (WRR) [18], Least Load, Least Connections, and Fastest Response algorithms [19]. For CSTgrid platform, in *ResourceAllocator*, we implemented, as failure-safe policy, the Dynamic Weighted Round-Robin (DWRR) [20] for load balancing. DWRR is a variant of WRR, in which the main merit of the algorithm is to minimize the frequency of detection. *ResourceAllocator*, calling the method to perform a DWRR, detects each host's load in the system at intervals and, following the detection of loads, a set of weights (the inverse ratio of host loads) is given to each host. The system allocates new jobs to each host according to this set of weights.

The resource tier, originally composed of a set of four web services working together to compose the CSTminer output, has been rewritten from scratch to permit the grid execution of Gaussian jobs. In the Gaussian grid just one resource is available to the user, which with appropriate scripting, is in charge of the code executions over a single node of the grid. The Gaussian resource is replicated on the grid (unlimited) hosts, providing each node with the appropriate script interface to the Gaussian03 executables. We will see the details of the grid interface to the Gaussian03 code in the next section, but it is worth noting that this scheme is absolutely portable to any QM code with similar computing workflow, by providing the appropriate script to the resource tier. Furthermore, this scheme for the replication of the resource(s) over the grid nodes simplify the jobs' "load-balancing", reduces the inter-node I/O and permits, in principle, to manage multi-QM jobs within the same scheme of reference.

3 Grid implementation of the code

The Gaussian grid application is composed of three layers each performing a given task: (1) a pre-processing tier; (2) a task execution tier and, finally (3) a data collector tier for the post-processing phase. The leading components of the Gaussian grid architecture are depicted in Fig. 2, where the first two stages are shown in detail.

The first step of the execution flow (in the pre-processing phase) is implemented via a proper web interface (Fig. 3) where a WS interacts with the end-user for the authentication, data input and job configuration. In a second step the pre-processing WS build up an execution table containing all the Gaussian03 input files that are generated taking into account the user data of the first step. The domain of this phase is the http server running the WS and all the data are locally stored and managed.

The second tier of the Gaussian grid application is a java class [21, references there in] responsible for the task generation for each of the Gaussian03 input files created by the pre-processing WS. The Gaussian grid java class, after contacting the grid manager via its daemon interface, spawn over the grid nodes the scripts for the Gaussian03 execution. This phase of the application is somewhat critical, since proper interaction of the java class with the Server Allocator determines the correct "load-balancing" of the Gaussian03 tasks over the whole computational grid. While the load balancing algorithm is able to cope with various execution queues to obtain the maximum node loads, QM jobs depend also on the specific input data. Then, the pre-processing tier plays a significant role also in the management of the task

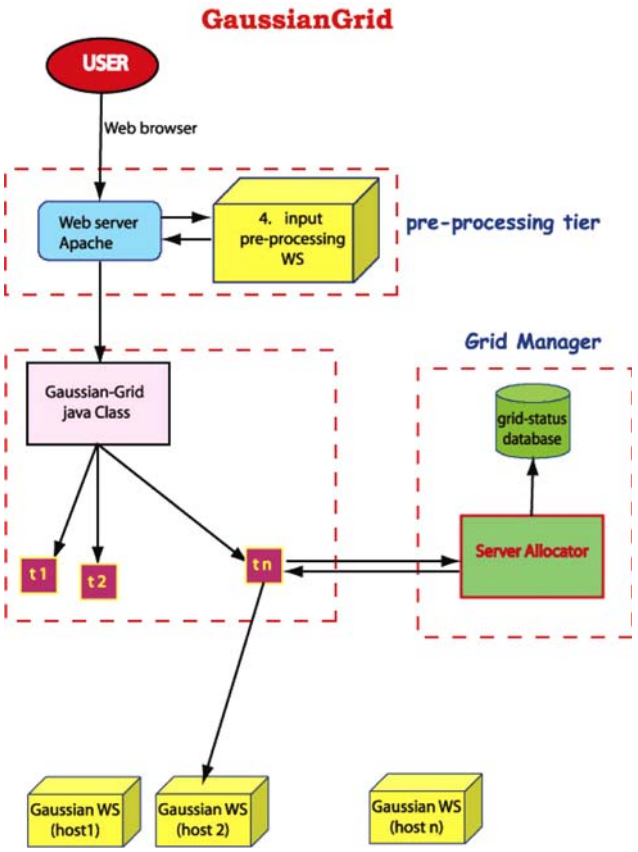

Fig. 2 The architecture of the Gaussian grid

execution over the grid, where a set of (as much as possible) similar input files should be generated in order to avoid unnecessary computing bottlenecks. To this end, in the pre-processing tier the Gaussian input files are generated keeping in mind these requirements, with equal "computational effort" (same number of mesh points to be computed, see next section for details) and with a fixed number of CPU's for each QM task that will be spawned by the master java-class over each node of the grid. Of course, in a future release of the Gaussian grid application a more sophisticated methodology for input generation should be implemented, and the whole "load-balancing" over the grid must be accomplished only by the Server Allocator which should be able to dynamically select the number of CPUs for each node as a function of the input data and host CPU availability. Nonetheless, as shown by the benchmarks summarized in the next section, already this preliminary implementation of the grid job management shows fairly good performances in the "load-balancing" of the Gaussian tasks over the grid nodes.

The last phase of the Gaussian grid workflow, the post-processing stage, has been intentionally presented with less detail to the reader. In fact, this part of the execution flow is the most customizable one by the site/user depending on the specific results to be extracted from the outputs of the various Gaussian03 tasks. In

Fig. 3 The Gaussian grid interface to the *GridInfo()* WS function

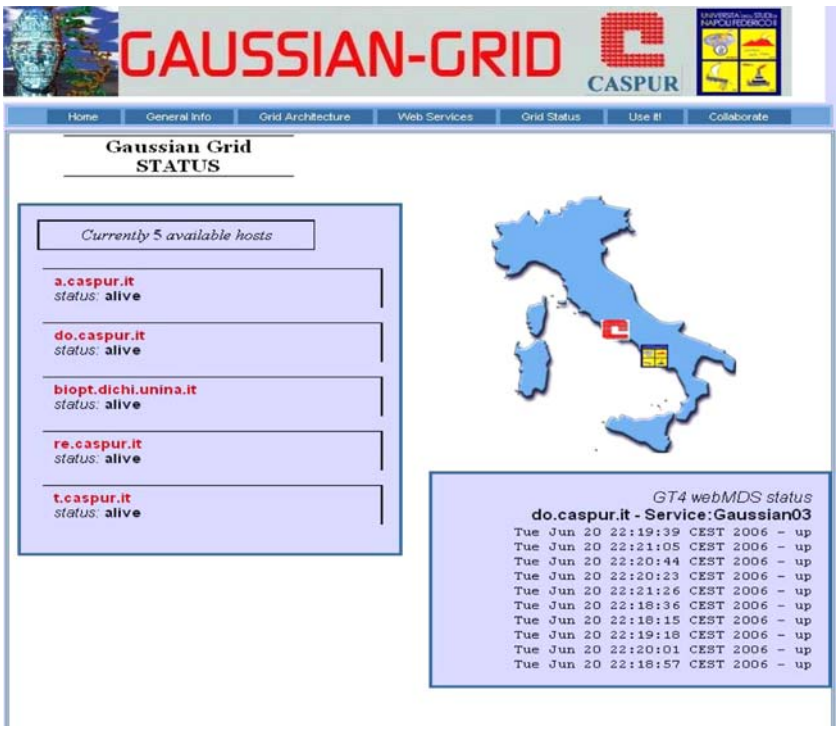

Table 1 HF/6-31G* with CPCM single-point energy calculation timings (single task, in seconds) and code efficiency (%)

Machine	Timing efficiency	4 × (Nproc = 1)	4 × (Nproc = 2)	2 × (Nproc = 4)	2 × 4(Nproc = 1)	4 × 2(Nproc = 1)	2 × 2(Nproc = 1) + 2 × 4(Nproc=1)
P2_1	Java (s)	21.4	15.3			22.9	22.7
	G03 (s)	20.3	13.7			20.1	20.2
	%Eff	94.8	89.5			87.8	90.0
P2_2	Java (s)	21.5	15.2			23.1	22.7
	G03 (s)	20.5	13.8			20.3	20.3
	%Eff	95.3	90.8			87.8	91.0
P4_1	Java (s)	22.3	15.5	13.3	22.2	23.1	22.4
	G03 (s)	21.4	14.5	12.1	20.5	21.1	20.6
	%Eff	96.0	93.5	91.0	92.3	91.3	92.0
P4_2	Java (s)	22.0	15.5	13.4	22.2	22.8	22.5
	G03 (s)	21.0	14.5	12.4	20.6	21.2	20.6
	%Eff	95.5	93.5	92.5	92.8	93.0	91.6
Overall	(s)	22.4	15.6	13.5	22.9	23.2	22.9
	%Eff(G03)	94.9	94.5	97.6	99.5	94.8	98.1
	%Eff(Java)	95.5	97.4	98.5	96.9	98.3	97.8

Data refer to bi-(P2_) and quad-processors (P4_) machine's grid running various combinations of Gaussian03 tasks at different Nproc values (see text for details)

our specific example, where the same QM calculation (method/basis set) was carried out on equally spaced set of Cartesian mesh points, the post-processing of the results was quite an easy task. In fact, each task returned one or more floating-point numbers with the value(s) of the molecular property to be computed over the mesh and the master class collected them on file so that a simple table of mesh point-property values was assembled for data visualization. Albeit simple, this post-processing scheme, clearly demonstrates the feasibility of the Gaussian grid workflow up to its last stage even for a very large number of tasks. Moreover, the last stage of the workflow is script based, that is, it is totally open to enhancements depending on the molecular property to be calculated over each Cartesian mesh point and on the final phase of data visualization which has not been taken into account in our experiment.

4 Distributed parallel test for benchmarking

The suite of tests we designed for the benchmarking and applicability of the Gaussian grid for real case QM calculations was based on simple (repetitive) tasks to be dispatched across the grid nodes. The reason for choosing such an approach is twofold. First, simple tasks can be executed in a limited amount of computing time thus permitting to single out any occurring bottleneck during the execution flow and second, simple tasks could be easily extended (typically adjusting only one parameter) to proceed with increasing execution time. Keeping this approach in mind, we have setup two sets of single point (SP) energy calculations over the Gaussian grid using the PCM(HF/6-31G*) and B3LYP/6-31G* methods on a mesh of varying x, y, z coordinates for the water molecule with a fixed step size for the increment (δ) of the ground state molecular configuration. Fixing the Nδ maximum extension of each x, y, z, coordinate, the 6N SP energy calculations were distributed over the Gaussian grid with various task combinations so that all the possible cases of G03 processes (and threads within) were evaluated. In particular, modifying the Nproc parameter of the G03 input file we were able to calculate the parallel code efficiency as result of the G03 process timing versus the dispatched execution time measured on the master java class which generated them. To this end, we report in Tables 1 and 2 the results of such a measurement in a test case where two AMD bi-processors and two AMD quadri-processors (2.6 GHz clock) were used for the Gaussian grid interconnected via a LAN network. The tables show the single task execution times (seconds) for different combinations of distributed processes in each column with a detailed timing from the G03 internal routines and from the master java class. Then, we calculated the code efficiency as the ratio of the G03 over the java class timing and we show them in each case of execution over the grid, so the user can easily evaluate the performance of the Gaussian code when it will be dispatched in such a distributed way. The results reported in Table 1 show that the G03 efficiency is slightly influenced by the grid execution with about a 10% performance reduction with respect to the "free" Gaussian code. These results are in line with the expected performance of the Gaussian

Table 2 B3LYP/6-31G(3*df*, 2*p*) single-point energy calculation timing (single task, in seconds), code efficiency (%) and speedup (SU)

		Overall best performance			Overall averaged performance		
Machine	Timing efficiency	4 × (Nproc = 1)	4 × (Nproc = 2)	2 × (Nproc = 4)	4 × (Nproc = 1)	4 × (Nproc = 2)	2 × (Nproc = 4)
P2_1	Java (s)	308.0	159.1		310.5	159.1	
	G03 (s)	306.9	158.0		308.9	158.0	
	%Eff	99.6	99.3		99.6	99.3	
P2_2	Java (s)	307.7	159.9		310.9	159.6	
	G03 (s)	306.6	159.6		309.8	158.5	
	%Eff	99.6	99.8		99.6	99.3	
P4_1	Java (s)	308.7	166.2	95.9	310.3	168.2	97.7
	G03 (s)	307.6	165.2	94.6	309.3	167.2	96.6
	%Eff	99.6	99.4	98.6	99.7	99.4	98.9
P4_2	Java (s)	308.4	165.2	96.0	309.6	164.3	96.7
	G03 (s)	307.2	164.1	94.9	308.5	163.3	95.3
	%Eff	99.6	99.3	98.8	99.6	99.4	98.6
Overall	(*s*)	308.8	166.3	96.1	311.0	168.3	97.8
	%Eff(Java)	99.6	95.7	99.8	99.5	94.5	98.9
	SU	1.00	1.86	3.21	1.00	1.85	3.18

Data refer to bi-(P2_) and quad-(P4_) processor machine's grid running various combinations of Gaussian03 tasks at different Nproc values (see text for details). Results refer to two data sets: best and averaged performance

executables when used for very fast calculations where the delay due to the grid dispatching (1–2 s depending on the number of processes spawned) strongly influences the overall elapsed time. Nonetheless, even in this limit case, the global efficiency of the G03 execution over the grid (measured as the overall timing of the master java class) is fairly good passing from about 96% (4 processes over 4 machines) up to ca. 98% (12 processes over 4 machines). This limited effect on the overall computing efficiency of the number of processes spawned over the grid suggests that the overhead due to the dispatching is almost constant and directly dependent on the number of nodes besides the number of processes. In fact, repeating the same tests but with larger G03 execution times (Table 2) for each single task, we see that the same overhead of 1–2 s is observed for each process but in this case the overall efficiency is now very close to 99%. Moreover, the internal speedup of the G03 code calculated with respect to the Nproc parameter, results almost identical to the "free" G03 code (about 3.2 over 4 CPUs), demonstrating that the grid execution of QM tasks could be implemented with a minimal overhead on the parallel code scalability.

This result, together with the high stability of the distributed system, is extremely encouraging for the application of the Gaussian grid in large-scale computations where hundreds of tasks should be managed to be executed on hundreds of CPUs. The extension to this huge computational size will be eventually limited only by the accessibility of large, general-purpose, computing grids, but the authors' opinion is that, after the present experiment, grid-aware QM codes are almost ready to be efficiently ported over a geographically distributed environment.

On these grounds, our implementation can be already considered the method of choice for all the so-called "embarrassingly parallel" applications in which message passing among the different platforms is not needed and the times for pre- and post-processing are negligible. In the field of computational models for molecular sciences we can mention fitting of potential energy surfaces obtained by quantum mechanical methods for generating effective potentials to be used in Monte Carlo (MC) or Molecular Dynamics (MD) simulations [22] and the computation of harmonic frequencies by finite difference of electronic energies and/or energy gradients [23]. Note that, thanks to the ongoing development of effective TD-DFT approaches and their analytical gradients both for isolated molecules and for condensed phases [24], the range of application of the above approaches is being extended to the evaluation of Franck–Condon factors and other vibronic couplings of paramount relevance in several branches of molecular spectroscopy [25].

More recently, effective codes have been developed for the evaluation of vibrational frequencies [26], averaged molecular properties [27,28], and thermodynamic functions [29] beyond the harmonic approximation by a second-order perturbative approach. Since the rate-determining step of the model is the evaluation of energy (third and semi-diagonal fourth) and property (first, second and, in some cases, third) derivatives by finite differences, also in this case our implementation is directly applicable. Note that, using a semi-classical approach

[30], the model can be used also to compute reaction rates including anharmonic and tunnelling effects [26]. In the same vein, reaction path analyses (IRC, QST2, etc.) can be effectively partitioned on different nodes on the grid.

In the case of condensed phases, straightforward applications involve the evaluation of energies and/or properties at different k points on different platforms for applications in the field of polymers [31–34] and surface sciences [35,36].

The situation is more involved for more demanding applications like, e.g., direct dynamical computations by methods rooted in the Born–Oppenheimer or the extended Lagrangian (CPMD [37,38], ADMP [39–43], …) approach. However, pending further developments, our implementation can already be of interest replacing a single long trajectory by a bunch of shorter trajectories on different nodes.

An interesting case of application of our distributed method of QM computation is the evaluation of Gaussian Type Orbitals (GTOs) and their derivatives over a 3D mesh (a cube). In particular, the cube evaluation technique has been widely used in the Single Centre Expansion (SCE) approach to the calculation of molecular properties in many electrons systems. One of the codes implementing such a method is the SCELib library [44] where the many-centre GTOs solution of the ground electronic state for a given molecular system, are computed over a 3D mesh and then expanded with respect to the molecular centre of mass to obtain a numerical SCE wave-function. The initial phase of the SCE procedure carried out in the *sphint* function of SCELib, the GTO evaluation over a 3D mesh, adopts substantially the same algorithm we implemented in our tests for the computation of a function over a spatial grid. Then, it would be interesting to compare the parallel measured performance of the *sphint* function with respect to the same kind of calculation carried out using our Gaussian grid but taking care to rescale the result for the code efficiency we measured in the above-reported benchmarks. In the graph of Fig. 4 we report the results of this comparison by superimposing the *sphint* parallel speedup over a shared-memory machine up to 32 CPUs (IBM p691) with the Gaussian grid performance at 75, 85 and 95% code efficiency of a grid composed by 8 nodes with 4 CPUs each. It is evident from the data shown that even at the lowest overall efficiency the distributed computation results more advantageous than a single shared-memory machine. Moreover, considering that the expected performance of the Gaussian grid for such a simple 3D mesh evaluation of a GTO wave-function will be well beyond the 95% in overall code efficiency, the reader could easily evaluate the benefits of the distributed grid solution we experimented in this paper with respect to traditional, locally executed, parallel binaries.

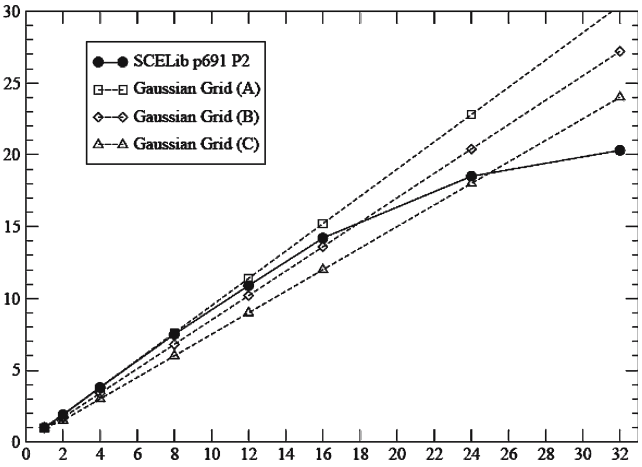

Fig. 4 The predicted scalability of the Gaussian grid for a GTO mapping over a 3D Cartesian grid. Data refer to 95% (*A*), 85% (*B*) and 75% (*C*) overall code efficiency (G03 internal × Java master class) compared to the measured performance of the SCELIb *sphint* function running on an IBM p691 configured as P2 (from Table 9 of Ref. [44])

5 Conclusions

In this paper we have presented an innovative solution to the execution of top class applications such those currently in use in quantum chemistry calculations. We had setup a grid environment and developed the necessary code sections to port one of the most used QM packages, G03, over a distributed, wide area network grid. This computational experiment, named the Gaussian grid, has proven to be an extremely stable computing environment when built up with advanced technologies such as service-oriented grid architectures and web services. The computational benefits of this solution have been proven to be even better than locally executed parallel applications and the Gaussian grid performance scalability is expected to be more cost-effective as the number of grid nodes increases. The application of the grid environment we had setup is not limited to the Gaussian package but the open architecture of the information technologies adopted in this experiment could be easily adapted to many parent computational codes. To this end, some examples of grid applications have been presented, including spatial mesh evaluation of function and their derivatives, generating effective potential for MC and MD simulations and reaction path analysis (IRC, QST) just to cite a few.

It is the authors' opinion that such distributed solutions for very large computing experiments in this scientific area could become of paramount importance provided a cooperative effort of chemistry- and technology-oriented researchers will be setup. Moreover, virtual laboratories with geographically distributed nodes (e.g. VILLAGE [45]) are becoming the mainstream solutions to the most demanding applications in many scientific sectors. The computational experiment presented in this paper is in line with those trends and we expect that, combining together other solutions based on grid technologies, the computational chemistry community will benefit most of the ongoing development of innovative models and of the corresponding computer codes.

References

1. Parr RG, Yang W (1989) Density-functional theory of atoms and molecules. Oxford University Press, New York
2. Koch W, Holthausen MC (2001) A chemist's guide to density functional theory, 2nd edn. Wiley-VCH, Weinheim
3. Marques MAL, Gross EKU (2004) Annu Rev Phys Chem 55:427
4. Tomasi J, Mennucci B, Cammi R (2005) Chem Rev 105:2999
5. Improta R, Barone V (2004) Chem Rev 104:1231
6. Barone V, Polimero A (2006) Phys Chem Chem Phys. DOI 10.1039/b607998a
7. Scalmani G, Barone V, Kudin KN, Pomelli CS, Scuseria GE, Frisch MJ (2004) Theor Chem Acc 111:90
8. Foster I, Kesselman C, Tuecke S (2001) Int J Supercomput Appl 15:3
9. Cerami E (2002) Web services. O'Really
10. Foster C, Kesselman C, Nick JM, Tuecke S (2002) The Physiology of the grid: an open grid services architecture for distributed system integration. http://www.globus.org/alliance/publications/papers/ogsa.pdf
11. Mignone F, Grillo G, Liuni S, Pesole G (2003) Nucl Acids Res 31:4639
12. Castrignano T, Canali A, Grillo G, Liuni S, Mignone F, Pesole G (2004) Nucl Acids Res 32(Web Server issue):W624-W627
13. Castrignano T, D'Onorio De Meo P, Grillo G, Liuni S, Mignone F, Talamo IG, Pesole G (2005) Bioinformatics, November 2
14. D'Onorio De Meo P, Carrabino D Sanna N, Castrignano T, Grillo G, Licciulli F, Liuni S, Re M, Mignone F, Pesole G (2006) FGCS (in press)
15. Foster I, Kesselman C (1997) Int J Supercomput Appl 11:115
16. Frisch MJ, et al. (2003) Gaussian 03, Revision C.02. Gaussian, Inc., Wallingford CT
17. Nagle J (1987) IEEE Trans Commun 35(4):435
18. Katevenis M, Sidiropoulos C, Courcoubetis C (1991) IEEE J Select Areas Commun 9:1265
19. River Stone Networks. Server load balancing: the key to a consistent customer experience. (2002) Technology white paper, No. 101:1–6, http://www.riverstonenet.co.jp/pdf/technology/whitepapers/SLBv5.PDF
20. Li D-C, Wu C, Chang FM (2005) Comp Oper Res 32:2129
21. http://java.sun.com and references therein
22. Chillemi G, Barone V, D'Angelo P, Mancini G, Persson I, Sanna N (2005) J Phys Chem B 109:9186
23. Carbonniere P, Begue D, Dargelos A, Pouchan C (2004) Chem Phys 300:41
24. Scalmani G, Frisch MJ, Mennucci B, Tomasi J, Cammi R, Barone V (2006) J Chem Phys 124:094107
25. Santoro F, Barone V, Improta R (2006) Angew Chem (in press)
26. Barone V (2005) J Chem Phys 122:014108
27. Barone V, Viglione R, (2005) J Chem Phys 123:234304
28. Barone V, Carbonniere P, Pouchan C (2005) J Chem Phys 122:224304
29. Barone V (2004) J Chem Phys 120:3059
30. Miller WH, Hernandez R, Handy NC, Jayatilaka D, Willets A (1990) Chem Phys Lett 172: 62
31. D'Amore M, Talartico G, Barone V (2006) J Am Chem Soc 128:1099
32. Improta R, Barone V, Kudin KN, Scuseria GE (2002) J Am Chem Soc 124:113
33. Improta R, Kudin KN, Scuseria GE, Barone V (2001) J Am Chem Soc 123:3311
34. Improta R, Barone V, Kudin KN, Scuseria GE (2001) J Chem Phys 114:2541
35. Cantele G, Trani F, Ninno D, Cossi M, Barone V (2006) J Phys C 18:2349
36. Festa G, Cossi M, Barone V, Cantele G, Ninno D, Iadonisi G (2005) J Chem Phys 122:184714
37. Car R, Parrinello M (1985) Phys Rev Lett 55:2471
38. Pavone M, Cimino P, De Angelis F, Barone V (2006) J Am Chem Soc 128:4338
39. Schlegel HB, et al. (2001) J Chem Phys 114:9758
40. Iyengar SS, et al. (2001) J Chem Phys 115:10291
41. Schlegel HB, et al. (2002) J Chem Phys 117:8694
42. Rega N, Brancato G, Barone V (2006) Chem Phys Lett 422:367
43. Brancato G, Rega N, Barone V (2006) J Chem Phys, (in press)
44. Sanna N, Morelli G (2004) Comput Phys Commun 162:51
45. VILLAGE, VIrtual Laboratory for Large-scale Applications in a Geographically-distributed Environment. http://www.lsdm.campusgrid.unina.it