Empathes: A general code for nudged elastic band transition states search

Marco Bertini[a], Francesco Ferrante[a,*], Dario Duca[a]

[a]*Dipartimento di Fisica e Chimica "Emilio Segrè" - Università degli Studi di Palermo, Viale delle Scienze Ed. 17, 90128 Palermo, Italy*

Abstract

An easy and flexible interface, `Empathes` (Extensible Minimum PATH EStimator), that allows to perform Nudged Elastic Band calculation for the determination of transition states is presented. The code is designed to be easily modified, in order to be associated with the user's preferred calculation software, even with those which implement composite approaches. In particular, the interfaces to Gaussian and Siesta programs are discussed in details, being the former only used for testing purpose, while the latter can be productively employed for transition states search with that commonly used density functional theory software for periodic calculations.

Keywords: NEB, `Siesta`, chemical reactions

PROGRAM SUMMARY

Program Title: `Empathes`

CPC Library link to program files: (to be added by Technical Editor)

Developer's repository link: https://github.com/marberti/empathes

Code Ocean capsule: (to be added by Technical Editor)

Licensing provisions: GPLv3

Programming language: Fortran 08

Nature of problem:

*Corresponding author

Email addresses: `marco.bertini@unipa.it` (Marco Bertini), `francesco.ferrante@unipa.it` (Francesco Ferrante), `dario.duca@unipa.it` (Dario Duca)

The search for the structure of transition states through computational methods, essentially based on Density Functional Theory, is of overwhelming importance for the determination of the elementary steps forming a reaction mechanisms. Allowing to develop basic knowledge, these investigations can be used to direct experimentalists towards a more efficient realization of chemical compounds synthetic processes. In cases where it is necessary to describe the reactive system through periodic calculations, which is very common in heterogeneous catalysis, this research must be done through the use of non-analytical methods.
Solution method:
In case of lacking of analytical procedures, the search for the transition states associated with the elementary stages that make up chemical reactions must take place through numerical methods. The Nudged Elastic Band (NEB) approach is, together with its variants, one of the most used for this purpose. In accordance with the NEB algorithm, a chain of geometric structures, generated by interpolating between the reactant and product geometries and joined by fictitious springs, is relaxed on the minimum energy path, allowing the association of the transition state to the maximum along this path. The NEB method involves the determination of molecular energies and forces acting on the nuclei of the system, which is generally carried out through a program for electronic structure calculation. The present code is a useful general interface.

1. Introduction

The Nudged Elastic Band (NEB) is a numerical method used for the discovery of the Minimum Energy Path (MEP) connecting two molecular geometries, generally the reagent and the product of a chemical reaction, along the Potential Energy Surface (PES). This method is largely used for studying transposition or migration processes, although it can be applied, at least in principle, to any kind of reaction. Knowledge of the MEP gives useful information about the transition state(s) (TS) and possible reaction intermediates. By knowing the

TS structure, useful energetic and kinetic information, like activation barriers and reaction rates, can be extrapolated.

The logic behind this method is pretty straightforward. Given the molecular geometries of reagent and product, an arbitrary number of intermediate geometries, called images, are interpolated between them. Those images are virtually connected one another by some fictitious springs, thus an *elastic band* of geometries is obtained, which will be gently *nudged* on the PES. This is done by iteratively calculating some appropriate forces acting on the images, and using them to optimize their geometries. The numerical process ends when the norms of the forces are below a given convergence threshold. If this threshold is chosen wisely, then the band will be an accurate estimation of the real MEP.

In the NEB method, as described in the work of Henkelman et al.,[1] each image i is optimized with respect to the total forces $\boldsymbol{F}_i^{tot}$

$$\boldsymbol{F}_i^{tot} = \boldsymbol{F}_{i\perp}^{PES} + \boldsymbol{F}_{i\|}^{el} \tag{1}$$

where $\boldsymbol{F}_{i\perp}^{PES}$ is the perpendicular component of the real atomic forces $\boldsymbol{F}_i^{PES}$ acting on the system, calculated by any computational chemistry software, and $\boldsymbol{F}_{i\|}^{el}$ is the parallel component of the elastic forces, computed by the Hooke's law. These forces are projected along the normalized tangent, $\hat{\boldsymbol{\tau}}_i = \boldsymbol{\tau}_i / \|\boldsymbol{\tau}_i\|$, on image i, with $\boldsymbol{\tau}_i$ computed as

$$\boldsymbol{\tau}_i = \begin{cases} \boldsymbol{\tau}_i^+ & \text{if } \; E_{i+1} > E_i > E_{i-1} \\ \boldsymbol{\tau}_i^- & \text{if } \; E_{i+1} < E_i < E_{i-1} \end{cases} \tag{2}$$

where E_i is the calculated energy of the image i, while $\boldsymbol{\tau}_i^+$ and $\boldsymbol{\tau}_i^-$ are defined as

$$\boldsymbol{\tau}_i^+ = \boldsymbol{R}_{i+1} - \boldsymbol{R}_i, \quad \text{and} \quad \boldsymbol{\tau}_i^- = \boldsymbol{R}_i - \boldsymbol{R}_{i-1} \tag{3}$$

being $\boldsymbol{R}_i$ the molecular geometry of the image i. If i is related to either an energy minimum or a maximum, the following alternative equation is used to compute $\boldsymbol{\tau}_i$

$$\boldsymbol{\tau}_i = \begin{cases} \boldsymbol{\tau}_i^+ \Delta E_i^{max} + \boldsymbol{\tau}_i^- \Delta E_i^{min} & \text{if } E_{i+1} > E_{i-1} \\ \boldsymbol{\tau}_i^+ \Delta E_i^{min} + \boldsymbol{\tau}_i^- \Delta E_i^{max} & \text{if } E_{i+1} < E_{i-1} \end{cases} \tag{4}$$

where

$$\begin{aligned} \Delta E_i^{max} &= \max(|E_{i+1} - E_i|, |E_{i-1} - E_i|) \\ \Delta E_i^{min} &= \min(|E_{i+1} - E_i|, |E_{i-1} - E_i|) \end{aligned} \tag{5}$$

Once determined $\hat{\boldsymbol{\tau}}_i$, the components of $\boldsymbol{F}_i^{tot}$ can be computed as

$$\boldsymbol{F}_{i\|}^{el} = k(\|\boldsymbol{R}_{i+1} - \boldsymbol{R}_i\| - \|\boldsymbol{R}_i - \boldsymbol{R}_{i-1}\|)\hat{\boldsymbol{\tau}}_i \tag{6}$$

$$\boldsymbol{F}_{i\perp}^{PES} = \boldsymbol{F}_i^{PES} - (\boldsymbol{F}_i^{PES} \cdot \hat{\boldsymbol{\tau}}_i)\hat{\boldsymbol{\tau}}_i \tag{7}$$

In this work we report the design of a new code, `Empathes`, for the application of NEB approach for transition states search. Although some commercial quantum chemistry software capable of applying the NEB procedure already exist (e.g. VASP [2]), the here proposed implementation of the method is general enough to be easily interfaced with virtually any program able to compute energy and atomic forces of a chemical system. At present, this software can interface `Gaussian` [3] and `Siesta`.[4] The former was chosen for testing purpose, since it has an implicit analytical method for TS discovery with which the TS found by the numerical NEB method can be compared. The latter was intended to supply an interface for production runs on periodic systems with `Siesta`, that can be used as an all-integrated alternative to the Flos library,[5] which needs external programs to generate NEB images.

2. Implementation Details

2.1. Modules Overview

The design behind this program involves communication between modules based on protected global variables, that is, accessible in read-only mode. When it is necessary to set one of these variables, this is done through a call to the appropriate subroutine. This prevents unintentional changes to their content.

Here a brief presentation of the modules follows:

`mod_input.f90`: deals with reading `Empathes` input file, and checks that all necessary arguments have been specified and are consistent.

`mod_geometry.f90`: contains information about the geometry of the images, as well as the linear interpolation procedure necessary to initialize them.

`mod_idpp.f90`: the Image Dependent Pair Potential (IDPP) method [6] can be used as an initial estimation of the reaction path, and is an excellent alternative to the simple linear interpolation. Within this module the subroutines are defined, which determine the value of the IDPP object function and its derivatives, to be used within a minimization method.

`mod_elastic.f90`: is the heart of the NEB method. This module collects the subroutines for the calculation of the tangents $\hat{\boldsymbol{\tau}}_i$ between images, as well as those for the decomposition of PES and elastic forces into $\boldsymbol{F}_{i\perp}^{PES}$ and $\boldsymbol{F}_{i\parallel}^{el}$, and for the determination of the total forces $\boldsymbol{F}_i^{tot}$ acting on each image.

`mod_pes.f90`: here are specified the subroutines that interface to computational chemistry software, necessary to get the energy E_i of the images and the atomic forces $\boldsymbol{F}_i^{PES}$ acting on them. These calculations can also be performed in parallel thanks to a specific subroutine that uses the MPI library.[7]

`mod_pes_data.f90`: set additional information for external calculations, such as the maximum number and convergence threshold of SCF cycles, as well as the names of auxiliary files.

`mod_pes_input_template.f90`: the data written inside the input blocks will be stored in specific structures defined here. This module defines subroutines to store the read data in the appropriate location, and to write it where necessary during the composition of the external program input files.

`mod_climbing.f90:` contains the procedures to perform the Climbing Image method (CI) [8] and the Descending Image (DI) methods, which is an analogue of the previous one that applies to the energy minima.

`mod_optimization.f90:` within this module are collected the subroutines that solve the minimization problem using the values of a function and its first derivatives in a given point (e.g. energies and PES forces, or the analogous quantities of the IDPP method). The Steepest Descent, the Fast Inertial Relaxation Engine (FIRE), the Broyden–Fletcher–Goldfarb–Shanno (BFGS) and the limited-memory BFGS are currently implemented, being the latter the default method thanks to its effectiveness.

`mod_output.f90:` all the subroutines that print information, such as geometries, forces, energy barriers, etc. are grouped here.

`mod_slave.f90:` defines a subroutine to put MPI slave processes in an idle state, waiting for the master to assign them a task.

`mod_utility.f90:` all global parameters and generic utility functions are collected in this module. The `get_field` subroutine is located here; how it is used for reading the output of the external programs will be detailed later.

`mod_c_utility.f90:` defines some Fortran subroutines to interface the C code contained in `c_utility.c`. This C module is intended to collect operating system specific functionalities (like changing the execution directory) in a portable way using standard OS libraries, making it possible, at least in principle, to compile this program on any linux system regardless of the compiler used.

[Figure 1 about here.]

[Figure 2 about here.]

2.2. Workflow

The flowcharts outlining how `Empathes` works are reported in Figure A.1 and A.2. Once read the input file, the geometries of the images that make up the elastic band are generated. This is done either through an interpolation technique, or through a direct reading of these geometries from a file. The latter case occurs mostly when a calculation needs to be restarted, but this feature can also be exploited to pass some handcrafted geometries to the program.

The NEB method is now applied. As equation (1) shows, the total forces are computed from the real PES forces. Obtaining $\boldsymbol{F}_i^{PES}$ and E_i is therefore the first step, as well as the critical part of the program and the one that takes almost all of the execution time. The present code can be seen as an automation of the NEB method in order to make it available with any computational chemistry program, interacting with it by writing its input, executing it, and reading its output. If the calculation has converged all the necessary information is read and stored, otherwise, if desired, the SCF convergence threshold is automatically increased up to a certain limit value, and the calculation is launched again.

The second step is the construction of $\boldsymbol{F}_i^{tot}$ from the information obtained above and from the equations of the NEB method. The third and last step is the optimization of the images with respect to their energy and $\boldsymbol{F}_i^{tot}$. This minimization problem has been solved in countless ways, and many algorithms are reported in the literature. The first optimizer to be used was the Steepest Descent, but due to its poor performance the FIRE [9, 10] and two flavours of BFGS [11] were implemented; the default method is the global limited-memory BFGS, which relaxes all image geometries at once.[12]

These three steps are repeated iteratively until the norm of $\boldsymbol{F}_i^{tot}$ is below a certain threshold for each i. Its value depends on the optimizer employed, but can also be set by the user. If all images converged, then a file containing the relaxed geometries is written, and the activation barriers for all TS found are printed in the `Empathes` output file.

2.3. Efficient IDPP forces derivation

The idea behind the Image Dependent Pair Potential approach is the generation of NEB images through linear interpolation on interatomic distances. This technique has a double advantage over the common linear interpolation performed on the coordinates. The first one is that the geometries of the images estimated this way follow a smoother path going from reagent to product. The second and most important one, is that it never produces geometries with atoms too close one another (a quite frequent case with linear interpolation), a situation that can make it hard for the external program to run successfully on that given image.

Since, given N atomic nuclei, there are generally more interatomic distances, $((N-1)N/2)$, than atomic coordinates, $(3N)$, the IDPP method cannot be applied as easily as the linear interpolation. The solution is to find the minimum of a properly defined objective function, S_i^{IDPP}; this problem is quite similar to the one solved by the NEB, with S_i^{IDPP} used in place of E_i and its derivatives instead of $\boldsymbol{F}_i^{PES}$, being the elastic part of the forces calculated exactly in the same way.

The equation set employed in the present implementation showed to define an efficient frame allowing to avoid system resources waste. Let's denote by d_{ij}^I and d_{ij}^F the interatomic distances between the i and j nuclei of the initial and final geometry, respectively. For each image k, the ideal interatomic distances can therefore be defined as

$$d_{ij}^k = d_{ij}^I + k(d_{ij}^F - d_{ij}^I)/(p+1) \tag{8}$$

where p is the total number of images. So, starting from an initial guess on the images geometries (generally by standard linear interpolation), a set of coordinates that produces interatomic distances d_{ij} as close as possible to d_{ij}^k is obtained. For this purpose, given that each image k contains N atoms, the IDPP objective function is defined as

$$S_k^{IDPP}(\boldsymbol{r}) = \sum_{i}^{N} \sum_{j>i}^{N} \omega(d_{ij}) \left(d_{ij}^k - d_{ij}\right)^2 \tag{9}$$

where $\boldsymbol{r} \equiv \{x_1, y_1, z_1, \ldots, z_N\}$ is the set of all the spatial coordinates, d_{ij} is the actual interatomic distance between i and j, and $\omega(d_{ij})$ is a weight function that prevents two nuclei from being too close each other

$$\omega(d_{ij}) = (d_{ij})^{-4} \tag{10}$$

By contracting all terms after the summations in (9) into a single term s_{ij}^k

$$s_{ij}^k = \omega(d_{ij}) \left(d_{ij}^k - d_{ij}\right)^2 \tag{11}$$

the forces acting on atom l, obtained by taking the negative derivative of S_k^{IDPP}, can be written as

$$\boldsymbol{F}_l^k = -\boldsymbol{\nabla}_l S_k^{IDPP} = -\boldsymbol{\nabla}_l \sum_i^N \sum_{j>i}^N s_{ij}^k \tag{12}$$

where only the s_{ij}^k terms having either i or j equal to l give not vanishing contributions. If a generic s_{ij}^k is derived with respect to a certain spatial variable α_r $(\alpha = x, y, z;\;\; r = i, j)$, we get

$$\frac{\partial}{\partial \alpha_r} s_{ij}^k = -4t \frac{\left(d_{ij}^k - d_{ij}\right)^2}{\left(d_{ij}\right)^6} - 2t \frac{\left(d_{ij}^k - d_{ij}\right)}{\left(d_{ij}\right)^5} \tag{13}$$

where

$$t = \begin{cases} +(x_i - x_j) & \text{if} \quad \alpha_r = x_i \\ -(x_i - x_j) & \text{if} \quad \alpha_r = x_j \\ +(y_i - y_j) & \text{if} \quad \alpha_r = y_i \\ -(y_i - y_j) & \text{if} \quad \alpha_r = y_j \\ +(z_i - z_j) & \text{if} \quad \alpha_r = z_i \\ -(z_i - z_j) & \text{if} \quad \alpha_r = z_j \end{cases} \tag{14}$$

Therefore, for a given atom l, the force component along α is given by the general equation

$$F_{l,\alpha}^k = \sum_{i=1}^{l-1} \frac{\partial}{\partial \alpha_i} s_{il}^k - \sum_{j=l+1}^{N} \frac{\partial}{\partial \alpha_l} s_{lj}^k \tag{15}$$

If these components are calculated sequentially starting from $l = 1$, then it can be argued that for a generic l the terms inside the first summation have already been calculated. In this way all the forces can be composed without wasting computing resources and system memory, that is by calculating once all the three spatial derivatives of all s_{ij}^k, and adding them in the right place in the final matrix that will contain the IDPP forces.

2.4. About Parallelization

As previously said most of the time is spent waiting for the external program to finish its calculations on the images. Since these computations are totally independent from each other, this is an embarrassingly parallel problem. It was therefore decided to make the code parallel using the MPI library, in order to support its use on computer clusters. However, the use of conditional compilation techniques also allows to build a serial executable suited for smaller systems.

Master and slaves is the parallelization paradigm employed, where the master performs almost all operations alone, with the exception of the calculations of PES energies and forces in which all processes contribute. Since the master is the only one who reads the `Empathes` input file, it must also initialize the slaves by broadcasting those information to them, so that they will be able to write the input files for the external program.

In order to carry out the PES computations in parallel, firstly the master broadcasts the images to the slaves since it is the only process that has the updated geometries. Then each process independently determines whether a given image belongs to it, and runs the PES computations on those that do. This determination does not require communication: it is as simple as taking the module between the image number and the number of processes, and comparing it to the process ID. Once each process executed all its PES computations, the inter-process communication begin. Slaves send the energies to the master via point to point communication. After that, master and slaves meet at the only MPI barrier present in the code, and finally also the forces are sent from slaves to master via point to point. It's worth to be noted that, with a single call to the `MPI_Send` subroutine, two information can be passed: one is either the energy or the force array itself, the other one is the image number to which those values refer, stored in the `tag` argument.

It is possible to run `Empathes` in parallel with any number of processes, although the best situation is achieved by setting it equal to the number of images used or a submultiple, so that each process has roughly the same workload.

It is to note, when the code is interfaced to quantum chemistry programs, which, like `Siesta`, use itself the MPI library, to run `Empathes` in parallel could not be a real advantage. In fact, a serial run of `Empathes` will allow the external program to continuously exploit all time long the available computational resources. However, there is the possibility to execute both `Empathes` and `Siesta` in parallel if the latter was compiled with OpenMP. Finally, if the end-user wants to employ an MPI version of `Siesta`, the #PESPROGRAMWITHMPI keyword must be specified in the input file.

3. Usage Examples

By executing `Empathes` with the "`-h`" option, usage instructions and more command line arguments will be printed. Some of them generate a template for the input file, that is a good starting point for the user since it lists and describes all the keyword that can be used to set the `Empathes` runtime behavior. Some keywords are mandatory, some are optional, and some are even mutually exclusive. They can be specified in any order, but an error message will follow if the same keyword is set more than once. Any keywords begin with a "`#`" character and could be followed by one or more arguments depending on its kind. A "`!`" at the beginning of the line, instead, states a comment that will be ignored by the program.

One of the main tasks of `Empathes` is to write the input files for the quantum chemistry software that will compute energy and atomic forces on images. From the `Empathes` point of view, these input files can be conceptually divided into two parts: one that varies from one image to another, and another that remains the same. Information to be written in the variable part must be somehow known by `Empathes`, so that it can adapt and write them appropriately. Among these information there are the geometries of the images, which change from one iteration to another; the SCF convergence threshold, so that `Empathes` can automatically lower it up to a limit value if the calculation does not reach it; and the maximum number of SCF cycles that will be performed by the external program, necessary to determine the convergence when using `Empathes` with some older versions of `Siesta`. These information must be specified by some appropriate keywords, like #START and #END for the starting and ending geometries, #SCFCONV and #SCFCYCLE for the SCF convergence threshold and maximum number of cycles, respectively. As for the part of the input that does not vary, this generally includes all the information related to the calculations that is intended to be performed on the various images. In this case, `Empathes` does not need to "understand" what will be written in this section of the input, but it will simply copy and paste these information, which are specified by means of the #PESINPUTTEMPLATE blocks.

The input files built by `Empathes` for `Gaussian` and `Siesta` are shown in the listings (1) and (2) reported in Appendix A, respectively. The first two characters of every line are here reported for explanatory reason, they are not printed in the real file. The first character indicates who prints that section of the input: a number means that it has been specified in the relative #PESINPUTTEMPLATE block, while an "E" indicates that `Empathes` itself prints it in that exact format. The second character "`|`", instead, is only a graphic separator, and is followed by what is actually written in the file. If `Gaussian` is used as the external program, the user may want to specify something after the geometry block, like a basis set or a pseudopotential. This can be done by specifying a "#PESINPUTTEMPLATE 3" block. If, instead, `Siesta` is used, more than two #PESINPUTTEMPLATE blocks are never needed, as it uses a free format for the input. In this case, however, it is worth noting that the geometry block written by `Empathes` must not contain chemical elements that refer to coordinates, but labels. For this reason, using `Siesta`, it is necessary to specify these labels by putting them as the fifth column in both the #START and #END geometries.

The following three examples of input files for `Empathes` will show how to execute a NEB calculation with `Gaussian` and with `Siesta`, as well as how to restart a NEB calculation if something wrong occurs.

3.1. Example 1 - `Gaussian`

Listing (3) shows an `Empathes` input file of use with `Gaussian`. Below the #START and #END keywords are specified, respectively, the coordinates of reagent and product in the *xyz* format. It has to be noticed that it is up to the user to ensure the consistency of the two geometries, since `Empathes` can only verify that the elements are inserted in the same order in the initial and final geometries, but it cannot do anything in the event that two lines referring to the same element have been accidentally swapped. The energies of those structures, needed to compute the tangents $\hat{\boldsymbol{\tau}}_i$, are given by #STARTENERGY and #ENDENERGY. Then the keywords that define the NEB computation itself begin: #OPTCYCLE specifies the maximum number of NEB iteration (a negative value indicates "until convergence is achieved"), #OPTCONV sets the threshold on the total force, #IDPP specifies that the IDPP interpolation must be used, #IMAGES tells the program how many images have to be generated, and "#CLIMBING 1" means that the CI-NEB will be performed only on "one" image that is an energy maximum. It is to say that, in the general case, "#CLIMBING n" will apply climbing up to the n highest energy maxima. The final part contains info on the PES calculation: #PESPROGRAM specifies the kind of program is intended to use, #PESEXEC contains the actual name of the executable, and the two #PESINPUTTEMPLATE blocks contain information to write in the input files for `Gaussian`.

3.2. Example 2 - `Siesta`

Listing (4) shows an `Empathes` input file of use with the `Siesta` code. Although the system in this example is the same as the previous one, the input file required to execute this NEB calculation is longer than that used for `Gaussian`, mostly due to the larger #PESINPUTTEMPLATE blocks. The first difference is a fifth additional column in the specification of both initial and final geometries. These numbers (but they can be anything, since they are stored as strings) are the labels related to every atom of the geometry. It's mandatory that the labels appear in the same order in both geometries, just like the atomic nuclei.

The next new keyword is #PESPROC, used to specify the number of processors on which the external program will run. This information is needed because a parallel run of `Siesta` is obtained via MPI. The #SCFCYCLE sets the number of SCF cycles that the external program will perform. #AUXINPUTFILES is used to specify how many and what auxiliary input files the external program needs. In this case only one file is needed, that is `*.psml`. Here the asterisk has the same function as in regular expressions, so all the files ending with `.psml` will be copied from the master directory into the working directories. The #AUX-OUTPUTFILES keyword has a similar behavior as the previous one. It can be used to specify how many and what auxiliary output files are to be stored, in order to reuse them in the next optimization iteration. Here the density matrix files are requested to be saved, that is all the files ending with `.DM`, since using them as the starting point for the next optimization iteration is a tremendous improvement in terms of computational time.

3.3. Example 3 - Restarting a Calculation

Suppose that the NEB calculation in listing (3) is launched and that for some reason it didn't end well. In the master directory there is a file called `lastgeom.bkp` containing, in order, the *xyz* molecular geometries of reagent, images and product, computed in the last well-ended optimization iteration. These can be used as the restarting point. In such a circumstance, it is preferable to copy `lastgeom.bkp` into another file, let's say `geometries.in`, and tell the program that the geometries it needs are located in this file.

Listing (5) shows an `Empathes` input file to restart a calculation. The #GE-OMETRIESFILE keyword is followed by the name of the file that stores the geometries. The first geometry is associated with the reagent, the last one with the product, an all the the intermediate geometries will be the corresponding images. When this keyword is specified, in order to avoid mistakes, the #START, #END, #IMAGES, and #IDPP keywords must not be present in the input file. The remaining of the input file is the same except for the new #CLIMB-INGQUICKSTART keyword, that is used to apply the Climbing Image method from the very first iteration.

4. Results

All the tests here reported has been executed using `Gaussian`09 as the external program, so that the TS obtained from the NEB method could be directly compared with those coming from its analytic method. Parallel performances were also investigated on a homogeneous cluster, where each node has two Intel Xeon E5-2690 processors for a total of 16 physical CPUs. Being the Intel Hyper-Threading technology enabled during the tests, there were 32 logical CPUs available on each node.

The first reaction studied was the simple $CH_2O \longrightarrow CHOH$ tautomerism. This system has been analyzed with the DFT B3LYP/cc-pVDZ method, 6 images were generated by IDPP interpolation, and the CI-NEB method was applied on one of them. Dynamic spring constants were used, and the FIRE optimization algorithm with a convergence threshold of $1.0 \cdot 10^{-3}$ $E_h/\text{Å}$ was set. Following this setting, convergence was achieved after 90 iterations. The TS found showed an energy of -114.370339 E_h, that is the same result obtained with the `Gaussian`09 analytic method. The running times related to parallel executions are reported in table (A.1). This table also contains a timing relative to an OpenMP run. This was the first form of parallelization implemented, but, being suitable only for computations on small systems, it has been removed, also considering the identical performance with MPI. In these runs, `Gaussian` was executed with 2 threads. The theoretical performance in this case is 1/6 of the serial time (i.e. 16.67%), that can be achieved only if each `Gaussian` calculation on each of the 6 images lasts exactly the same amount of time. Here a pretty close result of 19.03% is achieved in the best MPI run.

The second test was the acetone – propen-2-ol tautomerism. Once again the hydrogen hopping is involved, but this time in a slightly larger system. The calculation was carried out using the same conditions as in the previous case. The energies obtained for the TS from NEB and `Gaussian`09 computations were, respectively, -193.058873 E_h and -193.058869 E_h, so there is virtually no difference between the two results.

The last test was performed on a phosphoro-thionate cation, where the reaction
$CH_3{-}PO{-}SCH_3{}^+ \longrightarrow CH_3{-}PS{-}OCH_3{}^+$
involves the hopping of a methyl group from S to O.[13] This time B3LYP/6-311+G calculation was performed, and the NEB was set to use 8 images. Once again the analytic calculation and the estimation by NEB of the TS are in agreement, being the energies, respectively, -894.207182 E_h and -894.207170 E_h. Table (A.2) contains the computing times related to the parallel runs. For these tests the B3LYP/cc-pVDZ method was used, and `Gaussian` was executed with 4 threads. There being 8 images, the theoretical best time in this case is 12.50% of serial time, and in the best MPI run a very close 14.17% is obtained.

As regards a fairly more complex application, the authors very recently used `Empathes` interfaced with `Siesta` to successfully unravel the spillover mechanism of hydrogen atoms from Pd to graphene in a system formed by a Pd_4 cluster anchored on a C-vacancy of the graphene sheet. [14]

[Table 1 about here.]

[Table 2 about here.]

5. Extending the Interface

The interface of `Empathes` can be extended to other chemistry software. To do this, the user must edit the `mod_pes.f90` module only. Here he has to accomplish two tasks:

A to write at the end of the module three new subroutines that must:

1. write the input file for the external program
2. run it
3. read its output to get energy and atomic forces

B to modify the `get_pes_force` subroutine in three points, which can be find by searching the “`@end_user`” string:

1. setting a maximum SCF convergence threshold for the new program
2. inserting a check between the actual SCF threshold (stored in `conv_threshold`) and the maximum one
3. inserting the calls to the three previously written subroutines (point A), in that order.

The last two points require to specify a new `case` statement with the name of the new program, in which to insert the said logic.

The `get_pes_force` subroutine is used to obtain energy and atomic forces of the *i*-image. It sets some variables and then calls the three subroutines whose guidelines will be outlined shortly. It is to note that the arguments names are not casual, but are the same as the variables used in the `get_pes_force`: in this way the subroutine declarations and their calls are the same. After a successful implementation of an interface, to use `Empathes` with the new program it is mandatory to specify the #NEWPESPROGRAM keyword in its input, that disables some internal checks and makes mandatory the #SCFCONV and #SCFCYCLE keywords. The keyword #NEWPESPROGRAM exists in order to limit possible errors and simplify the implementation of a new interface.

5.1. Guideline for `write_progname_input` *Subroutine*

Arguments list

```
integer,      intent(IN)  :: i
real(DBL),    intent(IN)  :: conv_threshold
integer,      intent(IN)  :: fnumb_in
character(*), intent(OUT) :: fname_in
character(*), intent(OUT) :: fname_out
```

`i` is the image on which the computation will be performed; `conv_threshold` is the SCF convergence threshold to use; `fnumb_in` is the integer that must be associated to the input file through the `open` statement; `fname_in` and `fname_out` are respectively the names of the input and output files, that need to be set.

Useful global variables/subroutines/functions

```
integer           :: geom_len
real(DBL)         :: image_geom(i,j)
character(*)      :: element(i)
character(*)      :: elabel(i)
integer function  :: get_scfcycle()
subroutine        :: write_pes_it(fnumb,n)
```

`geom_len` is the geometry length: being N the number of nuclei, `geom_len`$=3N$; `image_geom(i,j)`, where $1 \leq$ `j` $\leq$ `geom_len`, is a matrix of real numbers, containing the j-coordinate of the i-image, following the order

$$x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N;$$

the array of strings `element(i)`, where $1 \leq$ `i` $\leq$ `geom_len`$/3$, contains the chemical symbols of the N elements; `elabel(i)`, where $1 \leq$ `i` $\leq$ `geom_len`$/3$, contains the labels specified in the fifth column of the geometry blocks; `get_scfcycle()` function returns the integer specified by #SCFCYCLE; `write_pes_it(fnumb,n)` subroutine writes the #PESINPUTTEMPLATE block `n` in an opened file associated with `fnumb`.

Subroutine body

Firstly the subroutine must set the names for input and output files. This can be achieved with

```
fname_in  = base_name//i_str//in_extension
fname_out = base_name//i_str//out_extension
```

where `base_name` is a global string defined in the current module, `i_str` is the transposition to characters of the integer `i`, and `in_extension` and `out_extension` are the input and output file extensions respectively.

After opening the input file `fname_in`, all the necessary information can be written in the appropriate format. Here the contents of the #PESINPUTTEMPLATE blocks can be pasted using the `write_pes_it(fnumb,n)` subroutine, while geometry, SCF cycles and SCF convergence threshold must be written manually using the above variables. If it does not do so by default, the keywords needed by the external program to calculate/write the atomic forces should be specified here. Once written, `fname_in` must be closed.

Arguments list

```
character(*), intent(IN)  :: fname_in
character(*), intent(IN)  :: fname_out !optional
logical,      intent(OUT) :: flag_conv !optional
```

`fname_in` is the input file name, that is generally required as an argument by the external program; `fname_out` is the output file name that is needed sometimes; `flag_conv` is used to store the convergence status: `.true.` if it has been achieved, `.false.` otherwise.

Useful global variables/subroutines/functions

```
character(*) :: pes_exec
integer      :: pes_proc
```

`pes_exec` is the executable name specified in #PESEXEC; while the integer `pes_proc` is the number of cores that the program will use, set by #PESPROC.

Subroutine body

This subroutine composes the command string to run the external program using the executable name stored in `pes_exec`, the file names, and the number of processors for the parallel run if they must be specified as command line arguments. The resulting string is executed by the operating system by passing it to the `execute_command_line()` subroutine. Sometimes, the exit code returned by the external program can be directly used to determine if convergence has been achieved or not. If this is the case, the value of `flag_conv` should be set here.

A final note on external programs that use MPI for parallelization. Since `Empathes` also use MPI, the external program can be launched via `mpirun` only if `Empathes` is run without `mpirun`. If a parallel run of `Empathes` is desired, then the external program must be launched using the MPI library subroutine `MPI_Comm_spawn`; this is not implemented in the current version of `Empathes`.

Arguments list

```
integer,       intent(IN)  :: i
integer,       intent(IN)  :: fnumb_out
character(*),  intent(IN)  :: fname_out
logical,       intent(OUT) :: flag_conv !optional
```

`i` is the image on which the computation has been performed; the integer `fnumb_out` must be used to `open` the output file named `fname_out`; `flag_conv` is used to store the convergence status: `.true.` if it has been achieved, `.false.` otherwise.

Useful global variables/subroutines/functions

```
real(DBL)         :: pes_energy(i)
real(DBL)         :: pes_forces(i,j)
integer function  :: get_scfcycle()
subroutine        :: get_field(str,field,n,err_n,err_msg)
```

`pes_energy(i)` is a vector of reals used to store the energy of the i-image; `pes_forces(i,j)`, where $1 \leq$ `j` $\leq$ `geom_len`, is a matrix of reals that stores the atomic force j of the i-image, in the order $F_{x_1}, F_{y_1}, F_{z_1}, F_{x_2}, \ldots, F_{z_N}$; `get_scfcycle()` function returns the integer specified by #SCFCYCLE; `get_field(str,field,n,err_n,err_msg)` subroutine can be used to extract the word `n` from the string `str`, that will be stored into `field`. If an error occurs, a non-zero value is returned in `err_n`, and an error message is set in `err_msg`.

Subroutine body

If the convergence status has been determined in the previous subroutine, the value of `flag_conv` can be used to establish whether the output should be read or not. Otherwise, the convergence status must be inferred by the output file itself, which therefore must be read in any case, and the `flag_conv` value must be set inside this subroutine.

This is the most critical subroutine to write, since the user must tailor its logic based on the format of the output file, which can differ sensibly from one program to another. For this purpose there is a subroutine, defined in the mod_utility.f90 module, called get_field() that extract a specific field from a string of characters. This is a pretty basic subroutine and, at present, the only supported separator between fields is the blank space (one or more characters). This example shows its use

```
str = "This␣␣␣is␣an␣␣example␣string"
call get_field(str,field,4,err_n,err_msg)
```

Here the fourth field from the string str (i.e. "example") is taken and saved in the field variable. It's a good practice to ensure that err_n is zero before using the content of field.

If the computation executed by the external program converged, then the user must locate the position of the energy and the atomic forces, and store them inside pes_energy(i) and pes_forces(i,:), respectively. It would be advisable to ensure that forces stored in pes_forces(i,:) are expressed in $E_h/Å$.

6. Conclusion

The Nudged Elastic Band approach frames a numerical method useful to estimate the minimum energy path connecting pairs of (energy-minimum) molecular structures, belonging to the same potential energy surface. The underneath NEB theory was here summarized along with the implementation design of the proposed code, Empathes. Some examples of using (employing different interface suit of programs, namely Gaussian and Siesta) and the corresponding input files with the following results got, were also given.

The main purpose of the Empathes code is to provide an implementation of the NEB method that is independent from any specific quantum chemistry software. By doing so, the end user would have the freedom to choose the software with which he usually works and to interface it with a NEB procedure.

Acknowledgement

This work was funded by the Italian Ministry of Economic Development (MISE) in the framework of the "Fondo per la Ricerca di Sistema Elettrico

- Piano Triennale 2019- 2021 - Progetto Sistemi di accumulo, compresi elettrochimico e power to gas, e relative interfacce con le reti".

References

[1] G. Henkelman, H. Jónsson, Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points, J. Chem. Phys. 113 (22) (2000) 9978–9985.

[2] G. Kresse, J. Furthmüller, Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set, Phys. Rev. B 54 (16) (1996) 11169.

[3] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, D. J. Fox, Gaussian˜09, Revision A.01, gaussian Inc. Wallingford CT (2009).

[4] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, The siesta method for ab initio order-n materials simulation, J. Phys.: Cond. Matt. 14 (11) (2002) 2745.

[5] FLOS: A lua library for linking with SIESTA, https://flos.readthedocs.io/.

[6] S. Smidstrup, A. Pedersen, K. Stokbro, H. Jónsson, Improved initial guess for minimum energy path calculations, J. Chem. Phys. 140 (21) (2014) 214106.

[7] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Version 3.1, 2015, https://www.mpi-forum.org/.

[8] G. Henkelman, B. P. Uberuaga, H. Jónsson, A climbing image nudged elastic band method for finding saddle points and minimum energy paths, J. Chem. Phys. 113 (22) (2000) 9901–9904.

[9] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, P. Gumbsch, Structural relaxation made simple, Phys. Rev. Lett. 97 (17) (2006) 170201.

[10] J. Guénolè, W. G. Nöhring, A. Vaid, F. Houllé, Z. Xie, A. Prakash, E. Bitzek, Assessment and optimization of the fast inertial relaxation engine (FIRE) for energy minimization in atomistic simulations and its implementation in lammps, Comput. Mater. Sci. 175 (2020) 109584.

[11] J. Nocedal, S. J. Wright, Numerical Optimization - Second Edition, Springer, 2006.

[12] D. Sheppard, R. Terrell, G. Henkelman, Optimization methods for finding minimum energy paths, J. Chem. Phys. 128 (2008) 134106.

[13] J. Barr, A. Bell, F. Ferrante, G. La Manna, J. Mundy, C. Timperley, M. Waters, P. Watts, Fragmentations and reactions of some isotopically labelled dimethyl methyl phosphono and trimethyl phosphoro thiolates and thionates studied by electrospray ionisation ion trap mass spectrometry, Int. J. Mass Spectrom. 244 (1) (2005) 29–40.

[14] F. Ferrante, A. Prestianni, M. Bertini, D. Duca, H_2 transformations on graphene supported palladium cluster: DFT-MD simulations and NEB calculations, Catalysts 10 (2020) 1306.

Appendix A. Listings

Listing 1: Input file for `Gaussian`, built by `Empathes`

```
1|%nproc=2
1|%mem=2GB
1|#p b3lyp/cc-pvdz
E|#scf(conver=8)
E|#scf(maxcycle=64)
E|#force test
E|
2|gaussian title
2|
2|0 1
E|O         0.000000       0.000000       0.000000
E|C        -1.209277       0.000000       0.000000
E|H        -1.896357       0.876338       0.000000
E|H        -1.558378      -1.162947       0.000000
E|
```

Listing 2: Input file for Siesta, built by `Empathes`

```
1|SystemName  lblneb
1|SystemLabel lblneb
1|NumberOfAtoms       4
1|AtomicCoordinatesFormat    Ang
1|%block AtomicCoordinatesAndAtomicSpecies
E|      0.000000      0.000000      0.000000 1
E|     -1.207081      0.000000      0.000000 2
E|     -1.868949      0.896768      0.000000 3
E|     -1.615736     -1.120896      0.000000 3
2|%endblock AtomicCoordinatesAndAtomicSpecies
2|NumberOfSpecies     3
2|%block ChemicalSpeciesLabel
2|  1  8  O
2|  2  6  C
2|  3  1  H
2|%endblock ChemicalSpeciesLabel
2|PAO.EnergyShift     0.005 Ry
2|PAO.SoftDefault     true
2|PAO.BasisType       split
2|PAO.BasisSize       DZP
2|LatticeConstant     1.0 Ang
2|%block LatticeParameters
2|30.0 30.0 30.0 90.0 90.0 90.0
2|%endblock LatticeParameters
2|xc.functional       GGA
2|xc.authors          PBE
2|SpinPolarized       true
2|MeshCutoff          450 Ry
2|DM.UseSaveDM        true
2|SolutionMethod      diagon
2|MD.TypeOfRun        CG
2|MD.NumCGsteps       0
E|DM.Tolerance        1.00E-04
E|MaxSCFIterations    64
E|WriteForces true
```

Listing 3: `Empathes` input to perform NEB/`Gaussian`

```
#START
4
O     0.000000    0.000000    0.000000
C    -1.203990   -0.000000    0.000000
H    -1.805419    0.945297    0.000000
H    -1.805419   -0.945297    0.000000

#END
4
O     0.000000    0.000000    0.000000
C    -1.317837   -0.000000    0.000000
H    -1.542714    1.108433    0.000000
H     0.296335   -0.928935    0.000000

#STARTENERGY -114.507640701
#ENDENERGY -114.423707780

#OPTCYCLE -1
#OPTCONV 1.0E-3
#IDPP
#IMAGES 6
#CLIMBING 1

#PESPROGRAM gaussian
#PESEXEC g09

#PESINPUTTEMPLATE 1
%nproc=2
%mem=2GB
#p b3lyp/cc-pvdz
#ENDPESINPUTTEMPLATE

#PESINPUTTEMPLATE 2
gaussian title

0 1
#ENDPESINPUTTEMPLATE
```

Listing 4: **Empathes** input to perform NEB/Siesta

```
#START
4
O        0.013513    0.000000   -0.000000  1
C       -1.204755   -0.000000    0.000000  2
H       -1.811774    0.966861    0.000000  3
H       -1.811774   -0.966861    0.000000  3

#END
4
O        0.005787   -0.000049    0.000000  1
C       -1.330060   -0.009745    0.000000  2
H       -1.547582    1.125306    0.000000  3
H        0.307075   -0.936793    0.000000  3

#STARTENERGY -643.692226
#ENDENERGY   -641.312969

#OPTCYCLE -1
#OPTCONV 1.0E-3
#IDPP
#IMAGES 6
#CLIMBING 1

#PESPROGRAM siesta
#PESEXEC siesta_psml
#PESPROC 16
#SCFCYCLE 200
#AUXINPUTFILES 1 *.psml
#AUXOUTPUTFILES 1 *.DM

#PESINPUTTEMPLATE 1
SystemName  lblneb
SystemLabel lblneb
NumberOfAtoms       4
AtomicCoordinatesFormat    Ang
%block AtomicCoordinatesAndAtomicSpecies
#ENDPESINPUTTEMPLATE

#PESINPUTTEMPLATE 2
%endblock AtomicCoordinatesAndAtomicSpecies
NumberOfSpecies     3
%block ChemicalSpeciesLabel
1  8  O
2  6  C
3  1  H
%endblock ChemicalSpeciesLabel
PAO.EnergyShift     0.005 Ry
PAO.SoftDefault     true
PAO.BasisType       split
PAO.BasisSize       DZP
LatticeConstant     1.0 Ang
%block LatticeParameters
30.0 30.0 30.0 90.0 90.0 90.0
%endblock LatticeParameters
xc.functional       GGA
xc.authors          PBE
```

```
SpinPolarized        true
MeshCutoff           450 Ry
DM.UseSaveDM         true
SolutionMethod       diagon
MD.TypeOfRun         CG
MD.NumCGsteps        0
#ENDPESINPUTTEMPLATE
```

Listing 5: Empathes input to restart an interrupted calculation

```
#GEOMETRIESFILE geometries.in

#STARTENERGY -114.507640701
#ENDENERGY -114.423707780

#OPTCYCLE -1
#OPTCONV 1.0E-3
#CLIMBING 1
#CLIMBINGQUICKSTART

#PESPROGRAM gaussian
#PESEXEC g09

#PESINPUTTEMPLATE 1
%nproc=2
%mem=2GB
#p b3lyp/cc-pvdz
#ENDPESINPUTTEMPLATE

#PESINPUTTEMPLATE 2
gaussian title

0 1
#ENDPESINPUTTEMPLATE
```

List of Figures

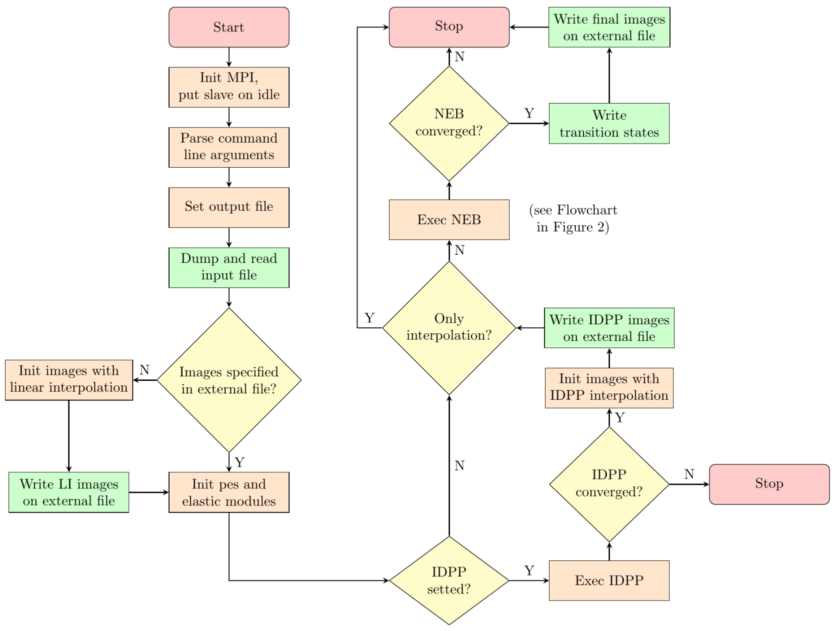
Start
Init MPI, put slave on idle
Parse command line arguments
Set output file
Dump and read input file
Images specified in external file?
N
Init images with linear interpolation
Write LI images on external file
Y
Init pes and elastic modules
IDPP setted?
Y
Exec IDPP
IDPP converged?
N
Stop
Y
Init images with IDPP interpolation
Write IDPP images on external file
N
Only interpolation?
Y
N
Exec NEB
(see Flowchart in Figure 2)
NEB converged?
N
Stop
Y
Write transition states
Write final images on external file

Figure A.1: Flowchart of the `Empathes` code.

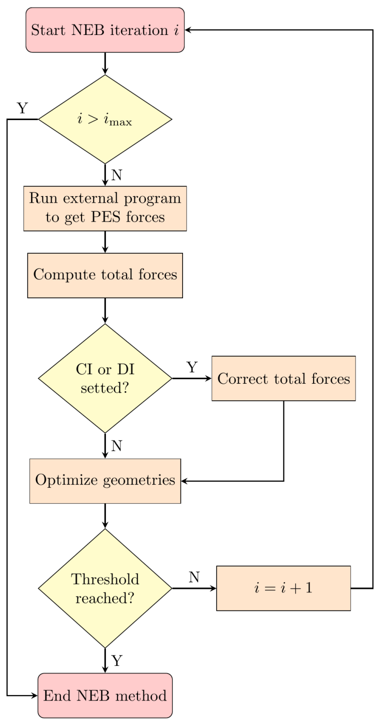

Figure A.2: Flowchart detailing the NEB algorithm.

List of Tables

Table A.1: H migration on formaldehyde.

Type[a]	c	N	c/N	U[b]	Time	%[c]
Serial	1	1	1	2	11m 56.282s	100.00
OpenMP	6	1	6	12	2m 16.541s	19.06
MPI	6	1	6	12	2m 16.317s	19.03
MPI	6	3	2	4	2m 25.274s	20.28
MPI	6	6	1	2	2m 49.560s	23.67
Theoretical best time					1m 59.404s	16.67

[a] `Gaussian`09 executed with t=2 threads. NEB executed with c processes on N nodes.

[b] Total number of CPUs used per node: $t \cdot c/N$.

[c] $\% = 100 \cdot \text{Time}/\text{Time}_{\text{Serial}}$.

Table A.2: CH_3 migration on phosphoro-thionate cation.

Type[a]	c	N	c/N	U[b]	Time	%[c]
Serial	1	1	1	4	2h 19m 13.107s	100.00
MPI	8	1	8	32	33m 38.119s	24.16
MPI	8	2	4	16	21m 48.204s	15.66
MPI	8	4	2	8	19m 43.837s	14.17
Theoretical best time					17m 24.138s	12.50

[a] `Gaussian`09 executed with t=4 threads. NEB executed with c processes on N nodes.

[b] Total number of CPUs used per node: $t \cdot c/N$.

[c] $\% = 100 \cdot \text{Time}/\text{Time}_{\text{Serial}}$.