Computer Physics Communications 214 (2017) 216–224

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

GPU Accelerated INtensities MPI (GAIN-MPI): A new method of computing Einstein-*A* coefficients☆

Ahmed F. Al-Refaie, Sergei N. Yurchenko, Jonathan Tennyson*

Department of Physics & Astronomy, University College London, Gower Street, London, WC1E 6BT, United Kingdom

ARTICLE INFO

Article history:
Received 21 June 2016
Received in revised form 26 December 2016
Accepted 15 January 2017
Available online 27 January 2017

Keywords:
Rotation-vibration spectra
Transition probabilities
Intensities

ABSTRACT

Calculating dipole transition intensities or the related Einstein A coefficients can dominate the computer usage for large line lists of transitions such as those being computed to model radiative transport through hot atmospheres. An algorithm for the efficient computation of line strengths is presented based on the use of the half-linestrength. This is implemented on GPUs that are shown to give up to a thousandfold speed-up compared to calculations on conventional computers. This algorithm is implemented in the program GAIN which was developed as part of the TROVE nuclear motion program, but can be adapted for use by other similar programs in a straightforward fashion.

Program summary
Program title: GAIN-MPI
Program Files doi: http://dx.doi.org/10.17632/4x75jsphc6.1
Licensing provisions: MIT licence.
Programming language: C++ 99, CUDA C and Fortran 95.
Nature of problem: Computation of linestrengths using GPU hardware
Solution method: Split the linestrength into smaller blocks and compute them in the GPU in parallel
Restrictions: The current version is restricted to separable rovibrational basis sets
Unusual features: Can be extended by user supplied concrete C++ classes for the MPI version
© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The calculation of rotation-vibration energy levels and wavefunctions for polyatomic molecules by direct solution of the Schrödinger equation for a given potential energy surface is now a standard procedure. There are a number of computer programs designed for these studies including Jensen's MORBID [1], Schwenke's VTET [2], DVR3D due to Tennyson et al. [3], WAVR4 due to Kozin et al. [4], TROVE due to Yurchenko et al. [5,6], ANGMOL developed by Pavlyuchko and co-workers [7,8], the MULTIMODE package developed by Carter, Bowman and co-workers [9–13], GENIUSH due to Matyus et al. [14] as well as DRV*6 and related codes by Mladenovic [15–17]. Carrington and co-workers [18–25] and others [26–30] have worked extensively on improving the methodology for obtaining variational solutions to these nuclear motion problems. In addition there are more specialized methods such as the code DOPI [31], which uses anharmonic force fields, and solution strategies based on the use of vibrational self-consistent field (VSCF) [32], vibrational perturbation theory [32] and multiconfiguration time-dependent Hartree (MCTDH) approach [33]. All these approaches are capable of yielding extensive and accurate sets of vibration–rotation energy levels and wavefunctions for a given potential energy surface.

Increasingly these wavefunctions are being used to compute transition intensities which, in favourable circumstances, can be obtained with an accuracy competitive with experiment [34–38]. Of particular concern here is the use of first principles nuclear motion methods to obtain extensive lists of Einstein-*A* coefficients. Such lists can be huge: for example recent studies on hot methane vibration–rotation spectra have produced extensive line lists [39–42] some of which contain many billions of transitions. Similarly large line lists have recently been produced as part of the ExoMol project [43] for NH_3 [44], PH_3 [45], H_2CO [46], H_2O_2 [47] and SO_3 [48]. Such large line lists are required for models of high temperature bodies [49]. Although computing the vibration–rotation wavefunctions necessary for calculating such line lists represents a challenge, the actual calculation of the transition dipoles or Einstein-*A* coefficients generally dominates the computer time usage. Indeed the amount of computer time required for

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail addresses: ahmed.al-refaie.12@ucl.ac.uk (A.F. Al-Refaie), s.yurchenko@ucl.ac.uk (S.N. Yurchenko), j.tennyson@ucl.ac.uk (J. Tennyson).

http://dx.doi.org/10.1016/j.cpc.2017.01.013
0010-4655/© 2017 Elsevier B.V. All rights reserved.

these computations has led to difficulties in completing these lists and proposals for drastically reducing them [50]. Here we show that graphical processor units (GPUs) can be deployed to break this bottleneck and greatly speed-up the computation of these huge line lists. Indeed the procedures developed here have proved key to successful computation of several of the line lists cited above [45–48]. The present paper gives the methodology used to significantly speed up the computation of transition dipoles (or linestrengths) and provides a code, GAIN-MPI (GPU Accelerated INtensities MPI), which provides an implementation of this methodology. Illustrations are given showing that this speed-up can indeed be very significant.

2. Method

The Einstein-*A* coefficient for a particular transition from the *initial* state *i* to the *final* state *f* is given by:

$$A_{if} = \frac{8\pi^4 \tilde{\nu}_{if}^3}{3h}(2J_i+1)\sum_{A=X,Y,Z}|\langle\Psi^f|\bar{\mu}_A|\Psi^i\rangle|^2, \tag{1}$$

where J_i is the rotational quantum number for the initial state, h is Planck's constant, $\tilde{\nu}_{if}$ is the transition wavenumber between upper state E_f and lower state E_i ($hc\,\tilde{\nu}_{if} = E_f - E_i$), Ψ^f and Ψ^i represent the eigenfunctions of the final and initial states respectively, $\bar{\mu}_A$ is the electronically averaged component of the dipole moment along the space-fixed (s) axis $A = X, Y, Z$ (see also Yurchenko et al. [51]). From this the absolute absorption intensity is determined by:

$$I(f \leftarrow i) = \frac{A_{if}}{8\pi c} g_{\rm ns}(2J_f+1)\frac{\exp\left(-\frac{E_i}{kT}\right)}{Q(T)\,\tilde{\nu}_{if}^2}\left[1-\exp\left(\frac{-c_2\tilde{\nu}_{if}}{T}\right)\right], \tag{2}$$

where c_2 is the second radiation constant, T the absolute temperature and $g_{\rm ns}$ is the nuclear spin statistical weight factor. Q is the partition function.

Computing the transition intensity requires computation of the Einstein-*A* coefficient given by Eq. (1), which is actually proportional to the line strength $S(f \leftarrow i)$ of a transition as given by [51]:

$$S(f \leftarrow i) = \sum_{\lambda',\lambda}\sum_{A=X,Y,Z}|\langle\Psi^f_{\lambda'}|\bar{\mu}_A|,\Psi^i_{\lambda}\rangle|^2 \tag{3}$$

where $\Psi^f_{\lambda'}$ and Ψ^i_{λ} are different ro-vibrational states with associated energies E' and E, respectively. It is often more practical to evaluate the linestrength using spherical tensor representations instead of the Cartesian representation of the dipole moment expressed in terms of the molecule-fixed (m) components as given by Bunker and Jensen [52]:

$$\mu_{\rm s}^{1,\sigma} = \sum_{\sigma'=-1}^{1}\left[D^{(1)}_{\sigma,\sigma'}(\phi,\theta,\chi)\right]^*\mu_{\rm m}^{1,\sigma'}, \tag{4}$$

where

$$\mu_{\rm m}^{1,\pm1} = \frac{\mp\mu_X - i\mu_Y}{\sqrt{2}}, \quad \mu_{\rm m}^{1,0} = \mu_Z. \tag{5}$$

Now Eq. (3) becomes:

$$S(f \leftarrow i) = \sum_{\lambda',\lambda}\sum_{\sigma=-1}^{1}\left|\langle\Psi^f_{\lambda'}|\bar{\mu}_s^{1,\sigma}|\Psi^i_{\lambda}\rangle\right|^2. \tag{6}$$

$$S(f \leftarrow i) = \sum_{\lambda',\lambda}\sum_{\sigma=-1}^{1}\left|\sum_{\sigma'=-1}^{1}\langle\Psi^f_{\lambda'}|[D^1_{\sigma,\sigma'}]^*\bar{\mu}_m^{1,\sigma'}|\Psi^i_{\lambda}\rangle\right|^2. \tag{7}$$

In the variational method, the wavefunctions Ψ^f and Ψ^i are generally represented as a linear combination of products of the vibrational and rotational basis functions obtained for a given combination of the rotational angular momentum J and irreducible representation Γ:

$$\Psi_n^{J,\Gamma} = \sum_{v,k} c_{v,k}^{J,\Gamma,n}\phi_v|J,k,m\rangle, \tag{8}$$

where ϕ_v is a vibrational basis function, $|J,k,m\rangle$ is the rigid rotor function where n is the 'running' number and $c_{v,k}^{J,\Gamma}$ are coefficients obtained by solving the nuclear motion problem. The molecular-fixed dipole moment components depend only on the vibrational coordinates and the $D^{(1)}_{\sigma,\sigma'}$ is dependent on the rotational (Euler) coordinates, therefore substituting the wavefunction of Eq. (8) into Eq. (7) and employing Clebsch–Gordan algebra to evaluate the rotational part gives [52]

$$S(f \leftarrow i) = g_{\rm ns}(2J'+1)(2J+1) \times \left|\sum_{v',k'}\sum_{v,k}\sum_{\sigma'=-1}^{1}(-1)^k[c_{v',k'}^{J',\Gamma',f}]^* c_{v,k}^{J,\Gamma,i} \times \begin{pmatrix} J & 1 & J' \\ k & \sigma' & -k' \end{pmatrix}\langle\phi_{v'}|\bar{\mu}_m^{1,\sigma'}|\phi_v\rangle\right|^2, \tag{9}$$

where $g_{\rm ns}$ is due to the nuclear spin degeneracy of the internal wavefunction. From the 3*j*-symbols the J selection rules for transitions are:

$$\Delta J = 0, \pm1, \quad J + J' \geq 1 \tag{10}$$

which are supplemented by the symmetry selection rules [52]

$$\Gamma' \otimes \Gamma \in \Gamma^* \tag{11}$$

where the product of symmetries Γ and Γ' must contain the symmetry of the dipole operator Γ^*. For individual matrix elements, the non-zero elements are constrained by the relation:

$$\Delta k = \sigma' = 0, \pm1, \tag{12}$$

which effectively eliminates the summation over σ' in Eq. (9).

There are two properties we can exploit. The first arises as all ro-vibrational wavefunctions in Eq. (8) possess the same 'primitive' vibrational basis functions ϕ_v. Therefore we can precompute and store all matrix elements of the dipole moment components $\langle\phi_{v'}|\bar{\mu_m}^{1,\sigma'}|\phi_v\rangle$, which will be referred to as $\mu_{v,v'}^{\sigma'}$. Secondly, the 3*j*-symbols can be effectively precomputed as well producing another matrix $F_{\Delta J,\Delta k}^{J,k}$. With this our linestrength simplifies to:

$$S(f \leftarrow i) = g_{\rm ns}(2J'+1)(2J+1) \times \left|\sum_{v',k'}\sum_{v,k}(-1)^k[c_{v',k'}^{J',\Gamma',f}]^* c_{v,k}^{J,\Gamma,i}F_{\Delta J,\Delta k}^{J,k}\mu_{v',v}^{\Delta k}\right|^2 \tag{13}$$

with the variational coefficients as the only varying quantity. However this is an $O(N^2)$ operation where N is the size of the ro-vibrational basis set, and evaluating it for each of the possibly billions of transitions with basis set sizes that could reach millions is cumbersome and inefficient. Therefore a two-step strategy is developed. First, the transition is projected onto the lower state wavefunction $\Psi_i^{J,\Gamma}$:

$$s_{v',k'}^{J',\Gamma'}(\leftarrow i) = \sum_{v,k} c_{v,K}^{J,\Gamma,i}(-1)^k F_{\Delta J,\Delta k}^{J,k}\mu_{v',v}^{\Delta k} \tag{14}$$

where $s_{v',k'}^{J',\Gamma}(\leftarrow i)$ is a vector that represents a 'half' transition from a lower state i to any state with J', Γ'. This is referred to as the *half linestrength* and is an $O(N^2)$ operation. A transition to any state

$\Psi_f^{J',\Gamma'}$ that satisfies Eqs. (10) and (11) can then be completed by performing a dot-product:

$$S(f \leftarrow i) = g_{\rm ns}(2J'+1)(2J+1)\left|\sum_{v',k'}[c_{v',k'}^{J',\Gamma',f}]^* s_{v',k'}^{J',\Gamma'}(\leftarrow i)\right|^2, \quad (15)$$

which is an $O(N)$ operation. This two step procedure has many advantages. When computing transitions up to a maximum wavenumber $\nu_{\max}$, the maximum energy of lower states $E_{\max}^i$ is determined by the relation:

$$E_{\max}^i = E_{\max}^f - \nu_{\max}, \quad (16)$$

where $E_{\max}^f$ is the maximum upper state energy. Because of this $N^i < N^f$, where N^i and N^f are the numbers of lower and upper states respectively. Additionally this has the consequence $N^t \gg N^i$ where N^t is the number of transitions. From this, the majority of the work is performed by the cheaper and faster dot product in Eq. (15) instead of the more expensive Eq. (14). This two step method is the methodology used in the variational nuclear motion code TROVE [5].

3. TROVE

TROVE's input is controlled by keywords and makes use of Stone's input parser [53]. Computing a transition requires modifying a TROVE input file to include an intensity block with the keywords given in Table 1. An example of the intensity part of the TROVE input file is given in Fig. 1. The dipole matrix elements $\mu_{v',v}^{\Delta k}$ are precomputed and stored in a checkpoint file, this is only done once in the entire TROVE pipeline and is read into memory for every transition run. At each run the matrix $F_{\Delta J,\Delta k}^{J,k}$ for a specified J range is computed and stored in memory, while the eigenvalues and quantum number assignments of states for all J in the range required are loaded and sorted by energy. The transition calculation occurs by looping through each lower state eigenvector $\Psi^{J,\Gamma,i}$ within the corresponding lower state energy range and computing all possible half-linestrengths. For each lower state J, Γ, i all half-linestrengths (see Eq. (13)) are computed for each J' and Γ within the upper energy range that satisfy the selection rules given by Eqs. (10) and (11). OpenMP is utilized during these half-linestrength computations independently working on k' and v' amongst available cores. The sum in Eq. (13) is restricted to $|k - k'| \leq 1$ and is also a subject of the expansion coefficient threshold condition

$$|c_{v,K}^{J,\Gamma,i}| \leq C_{\rm thresh},$$

with $C_{\rm thresh}$ of about $1 \times 10^{-12} - 1 \times 10^{-16}$. After this stage, a nested loop for upper state eigenvectors $\Psi^{J',\Gamma',f}$ is executed for energies and frequencies within the ranges requested, where the appropriate matrix $s_{v',k'}^{J',\Gamma'}(\leftarrow i)$ is selected for the dot product in Eq. (15). The dot-product is evaluated using the vendor specific BLAS [54] sub-routine library. In order to reduce the input/output (I/O) the required eigenvectors are batched into RAM. Since transitions from different lower states J, Γ, i are independent, the intensity calculations can be split into independent sub-ranges for $E_n^{(i)} \leq E_i \leq E_{n+1}^{(i)}$ and run in parallel over different nodes.

The main two factors that dictate the completion time of intensity calculations are (i) the total number of transitions (controlled via the keywords in Table 1) and (ii) the size of the basis set.

The size of a particular basis set d_J is given by:

$$d_J = (2J+1)\, d_{\rm vib}, \quad (17)$$

where $d_{\rm vib}$ is the size of the corresponding vibrational basis. Therefore as J increases the problem size increases. The time scaling

```
MEM 64 gb
SYMGROUP C2V(M)
(Other inputs)
INTENSITY
  absorption
  THRESH_INTES  1e-40
  THRESH_LINE   1e-40
  THRESH_COEFF  1e-40
  temperature   1000.0
  QSTAT        33314.25
  GNS          1.0  1.0  3.0  3.0
  ZPE  5773.228049563373
  selection  1 1 2 2
  J,  8,9
  freq-window  -0.001,  10000.0
  energy low   -0.001, 8000.00, upper  -0.00, 18000.0
END
(Other inputs)
```

Fig. 1. An example TROVE input with only relevant keywords (Stone's input parser [53] is used) for computing intensities for the H_2CO molecule.

for the half-linestrength is $O(d_J^2)$ and for the dot-product, $O(d_J)$. For large basis sets, the most significant bottleneck comes from the half-linestrength evaluation itself. For example, basis sets of around $\approx 10^6$ can take between 30 s to 10 min per lower state to complete with typically thousands of lower states per selected intensity run, especially for higher excitations. Table 2 shows typical half-linestrength times for a number of molecules as well as the total time spent performing this preprocessing step in the most demanding cases. Heavier molecules incur a greater burden at this step with SO_3 requiring over a month of wall-clock time.

Additionally there is an issue with the linestrength completion step in terms of the load-balancing; It is difficult to predict whether it is more efficient to use all cores for a single upper state filtering, eigenvector inflation and dot product computation or to perform multiple of these simultaneously with each given a single core.

However accelerators such as graphics processing units (GPU) are especially suited for highly parallel work such as the half-linestrength calculation as well as allowing for a more asynchronous work distribution between the CPU and GPU for parallel filtering, inflation of eigenvectors and the computation of dot products.

4. Using GPU architecture for intensity calculations

This section describes the terminology related to GPUs and features exploited in the program. The terminology and devices are based on the Compute Unified Device Architecture (CUDA). GPUs contain multiple streaming multiprocessors (SM) each a physical core used in execution, a register space and a small user-managed cache (32 KB) called *shared memory*. There is also a large on-board memory (1–12 GB) accessible by all SMs called *global memory*. A CPU issues commands to be executed by GPUs by calling routines known as *kernels*.

The hierarchy of memory is that registers are extremely fast (a few cycles) with the scope of a single thread, the shared memory is moderately fast (10–20 cycles) with the scope of the thread block and the global memory is slow (400 cycles) with the scope of the entire GPU. Each thread can read and write to the global memory and these persist across multiple kernels whilst shared and register memory are lost. Global memory reads can be improved by ensuring ordered access (*memory coalescing*).

It should be noted that the quoted term 'CUDA cores' is not analogous to the physical cores of a CPU. Instead they are more

Table 1
Keywords used in a TROVE input file (see Stone's input parser [53]).

Keyword	Comment
mem	Total memory in Gb
symgroup	Molecular symmetry group
intensity	Beginning of intensity block
absorption	Required keyword
thresh_line	Smallest linestrength to output
thresh_intens	Smallest absolute intensity to output
thresh_coeff	$=C_{\text{thresh}}$, used to skip below threshold $\lvert c_{v,K}^{J,\Gamma}\rvert$'s in Eq. (14)
temperature	Temperature of calculation
qstat	Partition function Q
gns	Nuclear statistical weight
selection	Symmetry selection rules
J	J range of calculation
zpe	Zero point energy
Freq-window	Frequency range of calculation
energy	*low*: E^i min,max – *upper*: E^f min,max
end	end of intensity block
( )	Comments

Table 2
Times in wallclock seconds for computing a single half-linestrength for H_2CO, PH_3 and SO_3 taken from the AYTY [46], SAlTY [45] and UYT2 [48] hot line lists calculations, respectively. The last row gives the total time in hours to compute half-linestrengths for the most dense $J \leftrightarrow J'$ with ≈4000 lower states.

J	H_2CO	PH_3	SO_3
10	3.46	13.84	13.84
20	7.43	29.72	131.87
30	11.30	45.20	263.01
40	16.32	65.28	381.05
50	21.25	85.01	512.19
60	25.89	103.57	625.15
70	30.07	120.27	828.25
Dense J total time (hours)	12.56	33.02	920.28

closely related to its SIMD (Single Instruction Multiple Data) width. These can be considered as 'coupled' cores to the physical core and can perform multiple mathematical operations simultaneously if the work given is predictable or *vectorizable*. A typical CPU has an SIMD width of around 128 bits or 4 floats. For a four core CPU this means the equivalent of 16 'CUDA' cores. GPUs rely on SIMD to a high degree to achieve the high number of 'CUDA cores'. A Tesla K20 GPU has 13 SMs with each having an SIMD width of 6144 bits or 192 floats giving a total CUDA core count of 2496. In a sense, there are only small number of true cores in a GPU but their ability to perform vectorizable computations is significantly augmented. The disadvantage is that work that contains conditions (e.g. if statements) are difficult to vectorize and the large SIMD width comes at the cost of lower clock speed and a smaller and weaker cache that means random memory access is slower and the user must manually manage a portion of the cache in order to achieve high performance.

All performance characteristics shown below are with the I/O time removed and using the eigenvectors and dipole moments from the AYTY linelist calculations [46] unless otherwise stated. The AYTY line list has vibrational basis set size of $d_{\text{vib}} = 7642$ and a maximum J at 70 giving a maximum basis set size of $d_J = 1\,077\,522$. The system used in measuring performance is the Emerald CfI cluster and comprises of 12 cores (two 2.50 GHz six-cores Intel Xeon E5-2640 processors) connected via NUMA with 8 nVidia Tesla M2090 GPUs attached (Fermi architecture).

5. Cache and reduce kernal

A naive kernel which we label as the *Baseline* was produced that implements Eq. (14) exactly as TROVE's implementation. The primitive basis function, 3j-symbols and dipole matrices are put into the GPU's global memory and each required eigenvector is transferred into the GPU's global memory before calculation. Fig. 2 shows the performance gain from the Baseline kernel against dimensions (different basis set sizes). Here we see that the GPU gives us a free speedup at smaller basis sets but becomes significantly less efficient at higher dimensions with only a 10% speedup at best. The reason for this is the sheer number of global reads for quantum numbers and coefficients required in Eqs. (14). Additionally, conditional statements are used to determine which dipole co-ordinate is accessed and to take advantage of the sparsity of eigenvectors to skip calculation. Both of these play to the disadvantage of the architecture of the GPU. Therefore a re-factoring in the overall methodology was required. Firstly, the k, v elements of the eigen-coefficients $c_{v,k}^{J,\Gamma,n}$ (i.e. how the ro-vibrational basis functions are arranged in the J, Γ basis set) can be rearranged by grouping them by the same v, i.e. effectively creating $(2J+1)$ blocks. Eq. (14) can also be decomposed into two further steps. The first is a caching step:

$$K_{v',k'}^{J',\Gamma',k,\Delta k} = \sum_{v} c_{v,k}^{J',\Gamma'} F_{\Delta J,\Delta k} \mu_{v',v}^{\Delta k}, \tag{18}$$

where $K_{v',k'}^{J',\Gamma',k,\Delta k}$ as the half-linestrength belonging to a specific k block is introduced for a specific Δk. The second is a 'reducing' step:

$$s_{k',v'}^{J',\Gamma'}(\leftarrow i) = \sum_{k}^{2J+1} (K_{v',k'}^{J',\Gamma',k,-1} + K_{v',k'}^{J',\Gamma',k,0} + K_{v',k'}^{J',\Gamma',k,1}). \tag{19}$$

In Eq. (18), all threads are guaranteed the same $c_{v,k}^{J,\Gamma}$ and $F_{\Delta J,\Delta k}$ making them easily cached into the faster shared memory during computation and the value for k is implicitly based on which block is being executed. Therefore the only global memory reads required are the v indices and $\mu_{v',v}^{\Delta k}$ matrices. The dipole reading is further coalesced as all threads will read around the same area of memory. The conditionals for determining the dipole co-ordinate were removed by transforming them into an easily calculated integer index and the threshold eigen-coefficient C_{thresh} is not applied, removing the sparsity advantage but improving the routine's vectorization.

This kernel will be referred to as the *CR Kernal* (Cache and Reduce Kernal). The kernel is called $(2J'+1)$ times in order to complete and each call is able to be performed simultaneously, to a degree, by using multiple streams as they are each independent of each other. Fig. 3 compares the efficiency of the *CR Kernal* to that of CPU and *Baseline* showing a substantial improvement with up to 30× speed-up from the CPU call time. The *CR Kernal* can complete the half-linestrength in less than a second for the largest basis set and speed-ups gained from this kernel increase with the growing basis set size. This comes entirely from the reduced global memory reads, utilization of the fast shared memory and data re-use.

5.1. Large dipole matrices

For the formaldehyde molecule used for the AYTY hot linelist, the size of the vibrational basis set is $d_{\text{vib}} = 7642$. This gives a dipole moment matrix of size $7462 \times 7462 \times 3$ which requires ≈1.2 GB using double precision. This fits easily into the M2090 or K20 GPU's global memory. However the linelists computed for molecules such as PH_3 and SO_3 require d_{vib} of 14386 and 15948 respectively. For the PH_3 case this gives a dipole of memory size 4.6 GB which only barely fits into the M2090 memory. However the SO_3 calculations require ≈5.8 GB to store which is infeasible both for the M2090 and K20 GPUs. The easiest solution would be to utilize the larger memory K40 and K80 GPUs to perform the computation but this would limit larger and heavier molecules to specific models of nVidia GPUs. Another possible solution (adopted here) that provides flexibility is to partition the dipole moment

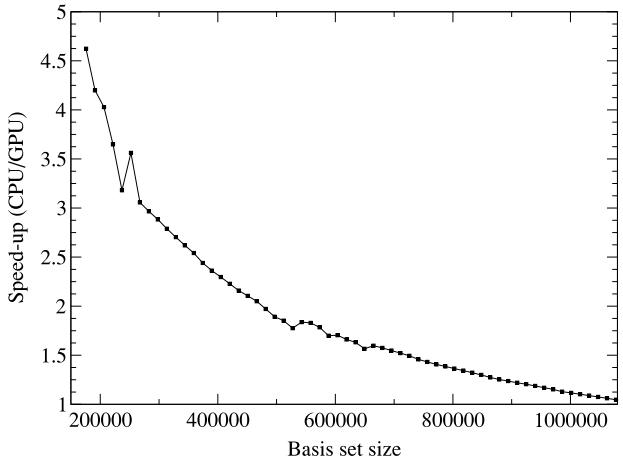

Fig. 2. A plot showing the speedup (CPU time / GPU time) of calculating the half-linestrength against basis set dimension when directly implementing Eq. (14) to CUDA.

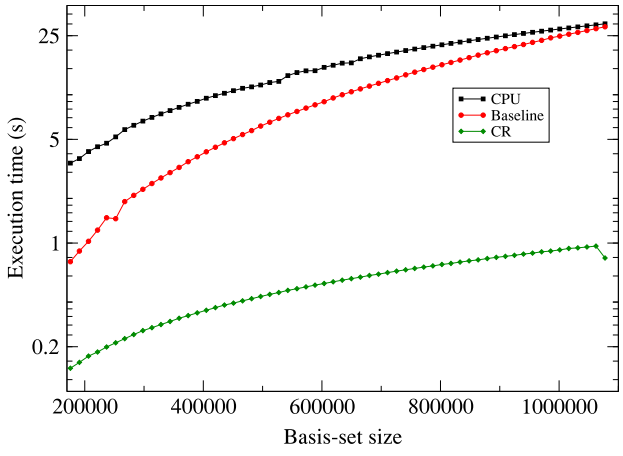

Fig. 3. Average half-linestrength call time in seconds on a logarithmic scale against the CPU (TROVE), the Baseline kernel and the Cache and Reduce (CR) kernel.

matrix into blocks that can fit into GPU memory and call a modified version of the CR kernel for each dipole moment block. The dipole matrix elements depend on v and v'. Each thread requires all v' to complete a particular k' and $\phi_{v'}$, the matrix is therefore partitioned by v into p number of blocks. The structure of Ψ' is such that each v' of the dipole matrix is applicable to every k' block and is shown in Fig. 4. A matrix block is transferred into the GPU and calls a block-CR kernel for each k. The block-CR kernel is almost exactly the same as the CR kernel only it spawns threads for v which exist in the matrix block. This is repeated for each matrix block in order to complete the half-linestrength.

Fig. 5 shows how the speedup gain from using this strategy varies over differing blocking (p) values. Overall, there is a performance reduction with increasing block-size with a 2× reduction for $p = 2$ and 3× for $p = 3$. This makes sense as we effectively need to call the kernel p multiple times to complete the half-linestrength as given by block-CR. Such blocking method is more beneficial for more difficult molecules such as SO_3. Fig. 6 uses the basis sets and vectors from the SO_3 hot linelist obtained by Underwood et al. [48]. As the dipole matrix elements are too large to fit in the M2090 memory, only partitioned dipoles are shown. Here the speed-up given is significant and the change in speed-up between p values varies by only ≈20%–30%. This is because of the basis set that can easily saturate the GPU with work even after splitting.

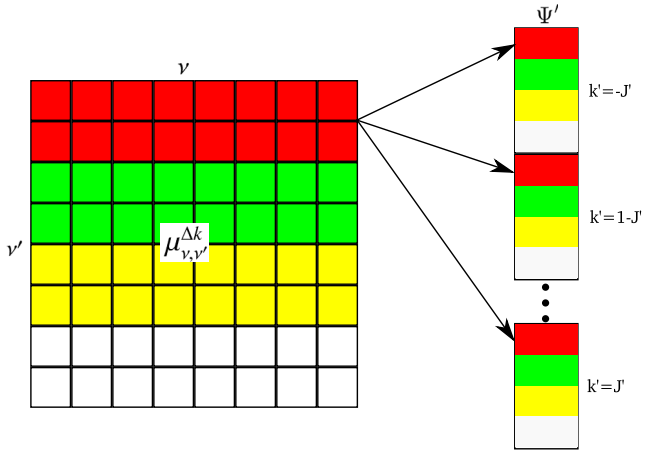

Fig. 4. A visual representation of blocking the dipole matrix elements with $p = 4$, the colours and arrows show how each matrix block relates the k-blocks in state Ψ'.

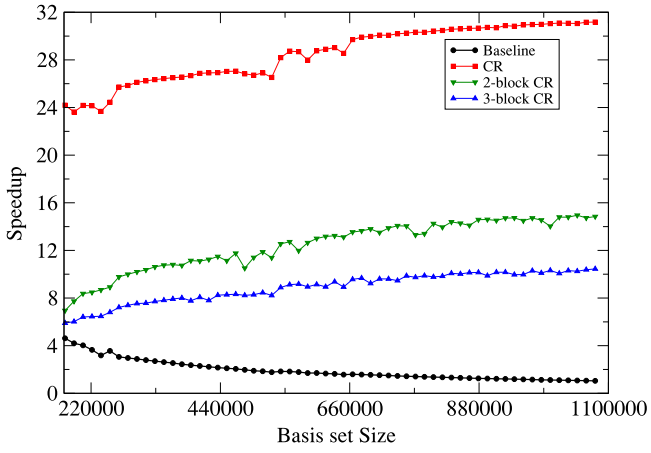

Fig. 5. Performance characteristics of the CR kernel with varying values for the blocking parameter p.

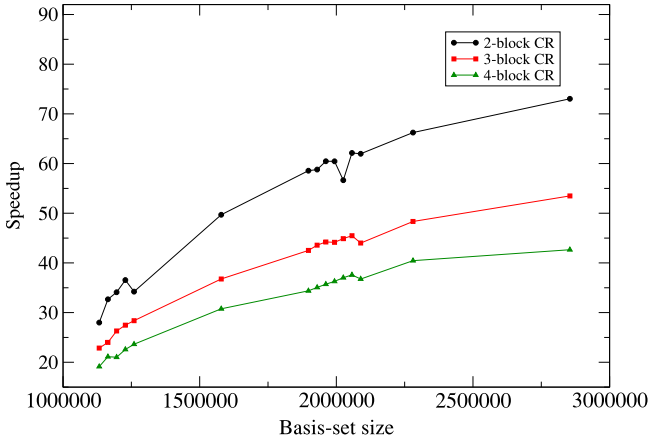

Fig. 6. Performance characteristics of the CR kernel with varying values for p using basis sets from the SO_3 calculations.

Overall, the half-linestrength calculations with the CR kernel are up to 70× faster than that with the CPU-only version and are especially suited for more difficult molecules, as the limitation of the vibrational basis set on the linelist completion time is reduced. This may encourage the use of larger basis sets which provide more accurate line positions via improved convergence of energies and

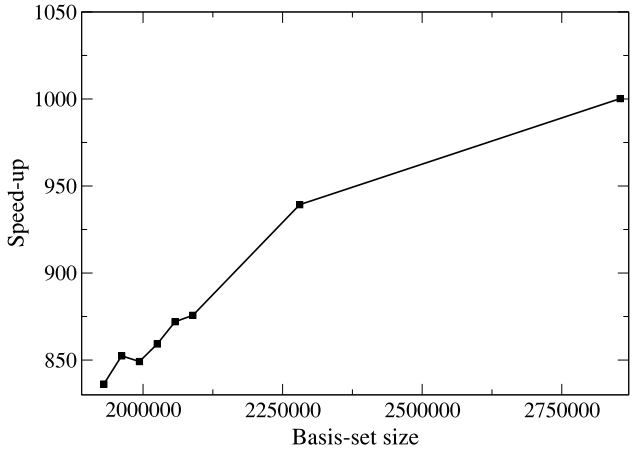

Fig. 7. Speed-up (CPU time / GPU time) achieved with the non-blocking CR on a K80 for the SO_3 linelist calculations.

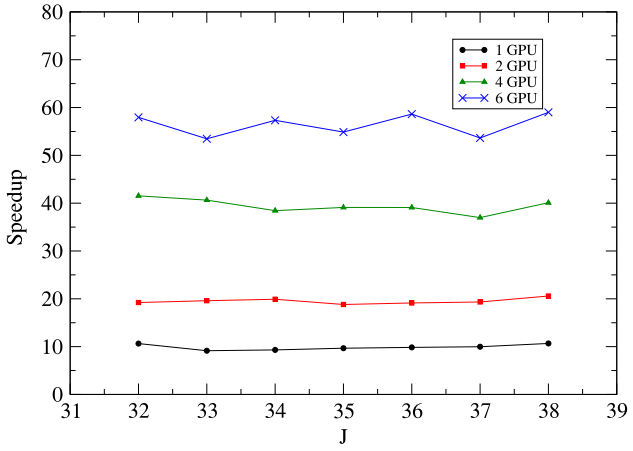

Fig. 8. 'Effective' linestrength performance increase with varying GPU setups for GAIN using the AYTY H_2O_2 line list. The speed ups are in comparison to TROVE with IO time removed.

larger wavenumber and energy (and thus temperature) ranges. Indeed this has already been done for methane where the use of GAIN has allowed the computation of a significantly extended line list [42].

In order to assess the efficiency of the blocking CR, it is worth comparing to an ideal situation where entire dipole matrix can fit into memory for the case of large SO_3 linelist calculations. Towards this we have acquired access to a K80 GPU with an identical setup to the M2090. The 12 GB of memory provided by the K80 gives us the ability to assess how the non-blocking CR compares to the blocking CR. Fig. 7 shows the speedup with the non-blocking CR kernel on the K80: For the largest cases presented, we can achieve a substantial performance increase with a 1000× speed-up comparing to the pure CPU TROVE calculations and a 10× increase from the 2-block CR. The latter is attributed to the reduced number of CR calls and the lack of stalling due to dipole matrix transfers. However the scaling of the algorithm is consistent for both with only a ≈50% increase in execution time when the basis set size is doubled compared to the CPU version which increases by ≈100%.

6. GAIN

GPU **A**ccelerated **IN**tensities (GAIN) is a set of functions that compute transitions rapidly using the CR kernels and the cuBLAS [55] implementation of the dot-product. The usage of the CR and non-blocking CR kernels is determined automatically by collecting GPU data and the dipole size at run-time. Additionally the code is asynchronous and allows for the CPU to work whilst the GPU is computing. GAIN is also compatible with OpenMP, by passing in the total number of cores, it can perform up to 10 dot products in parallel on a single GPU. It also has multi GPU support and will detect available GPU sockets and distribute them evenly across all cores. This can effectively speed up the dot-product step. Fig. 8 shows the 'effective' speed-up of the dot product performance on the AYTY line list. As a single linestrength completion time does not change for each GPU, it is not a true 'algorithmic' speed-up. However with N GPUs we can compute $10 \times N$ linestrengths simultaneously thus 'effectively' increasing throughput by $10 \times N$.

GAIN has been successfully integrated into TROVE with very few modifications to the overall intensity calculation subroutine. It should be straightforward to implement GAIN in other codes that use uncoupled vibration–rotation basis sets of the form given by Eq. (8). However, GAIN will require some adaptation for use with codes which employ coupled vibration–rotation basis sets, which are necessary to allow for the correct treatment of molecules whose vibrational motion can go from bent to linear geometries [56].

7. GAIN-MPI

GAIN-MPI is an extension of GAIN and is a hybrid OpenMP + MPI + CUDA C code that allows for the usage of a higher number of GPUs compared to GAINs single node setups. It allows for the mass production of transitions. In its default form, it operates on TROVE wavefunctions and utilizes the same input files as TROVE, more specifically the intensity input blocks seen in Fig. 1 and keywords in Table 1. GAIN-MPI supports all symmetries that TROVE supports, this is because it uses TROVE's symmetry FORTRAN files. Therefore upgrading to newer symmetries implemented in TROVE is a simple case of replacing the relevant FORTRAN files in GAIN.

Each process will read the input, load basis sets, states and if necessary, split the dipole moment matrix into the necessary number of blocks to fit into the GPU, after which the eigenvectors are distributed to a particular rank through the relation:

$$\text{Rank} = i \bmod N_{\text{procs}}, \tag{20}$$

where i is a state counting number and N_{procs} is the total number of MPI processes. Eigenvectors are cached into RAM until the memory for each process is exhausted, after which all further eigenvectors can be accessed from storage. Each process will read an initial state and determine whether it satisfies the filtering rules given by the input file and whether the state belongs to that particular rank. The process to which the state belongs performs the necessary half-linestrength calculations and broadcasts the results to all processes. Each process then loops through all states and performs the dot-products on those that satisfy the energy thresholds, selection rules and Eq. (20). Fig. 9 visually describes this process. This approach to distributing states and eigenvectors ensures that all ranks perform work within a given wavenumber range. Additionally, with enough MPI processes, I/O reads can be eliminated. Fig. 10 shows completion time for the PH_3 linelist calculations at $J = 20$. Each process has 6 GB of memory and the required total memory to store all eigenvectors is ≈140 GB. When 30 processes are utilized, I/O is completely eliminated and completion time takes less than an hour. A sample output is given in Table 3.

Fig. 9. Flow chart depicting GAIN execution. *s_id* refers to the state running number, *n_procs* the total number of MPI processes and *rank* the current MPI processes rank.

7.1. Usage

GAIN-MPI relies on being supplied TROVE checkpoint files to perform calculations. These checkpoint files store various outputs that can be reused in the TROVE pipeline including the dipole matrix elements and eigenpairs. The ones relevant to GAIN are described in Table 4. The checkpoint files required for all intensity calculations are the dipole matrix elements and the $J = 0$ eigen checkpoint files.

All J and Γ specified in an input file require their relevant 'descr', 'vectors' and 'quanta' checkpoint files in order to run the calculation. Included in the supplementary material are checkpoint files based on the H_2CO AYTY [46] linelist up to $J = 5$ with (see Appendix A). This range includes all bands available in the HITRAN database [57] for comparison. The line positions of the transitions will be of lower quality than those given in the AYTY line list as a significantly reduced basis set and operator expansion is used in order to reduce the size of the checkpoint files. Additionally included are the documentation and the input files needed to regenerate the checkpoint files supplied and the intensities using TROVE. Note that TROVE itself is freely available from the CCPForge program repository (https://ccpforge.cse.rl.ac.uk/).

Running GAIN-MPI without arguments provides the help output:

```
Usage: ./GAIN-MPI_KEPLER.x <input file> [options]
Options:
-h,--help    Show this help message
-o,--output FILENAME  Output linestrengths to
  files with format FILENAME__[MPI rank]__.out
-i,--compute-intensity  Compute absolute
  intensities in cm/molecule
-f,--full-linestrength  Output all linestrength
  components
```

The output is piped to stdout unless the `-o` flag is set in which case it is output to a file for each MPI rank n with the additional suffix `__n__.out`. This is used if the MPI library does not produce thread-safe output. Additionally the absolute intensities in cm/molecule can be computed from the Einstein-A coefficients if an additional `-i` argument is supplied and is concatenated after the default output. Lastly the full linestrengths for the transition can also be output if the argument `-f` is supplied and will be the very last output in the line.

Table 3
A sample default output produced by GAIN-MPI.

	v_{if}	n_f	J_f	Γ_f		n_i	J_i	Γ_i	A_{fi}
#	1174.246109	1	1	1	←	1	1	2	3.56497232E−01
#	1259.379381	2	1	1	←	1	1	2	1.39442514E+00
#	1498.358225	2	0	1	←	1	1	2	8.78353658E−01
#	1744.075964	3	0	1	←	1	1	2	3.53440984E+01
#	2423.803319	3	1	1	←	1	1	2	4.69999134E−05
#	2494.949157	5	0	1	←	1	1	2	5.71096415E−01
#	2674.487145	4	1	1	←	1	1	2	2.35255805E−01
#	2781.132468	6	0	1	←	1	1	2	5.86996598E+01
#	2852.641762	6	1	1	←	1	1	2	4.42291961E+01
#	2914.123444	7	1	1	←	1	1	2	2.37305983E−02

v_{if}: Transition wavenumber in cm^{-1}
n_f: Upper state counting number
J_f: Upper state rotational excitation J
Γ_f: Upper state symmetry number
n_i: Lower state counting number
J_i: Lower state rotational excitation J
Γ_i: Lower state symmetry number
A_{fi}: Einstein-A coefficient in s^{-1}.

Table 4
Checkpoint files.

Checkpoint file	Comment
j0_extfield.chk	Dipole moment matrix elements
j0eigen_descr$J$_Γ.chk	Contains eigenvalues for states with J and symmetry Γ
j0eigen_vectors$J$_Γ.chk	Contains eigenvectors for states with J and symmetry Γ
j0eigen_quanta$J$.chk	Describes the basis set for states with J

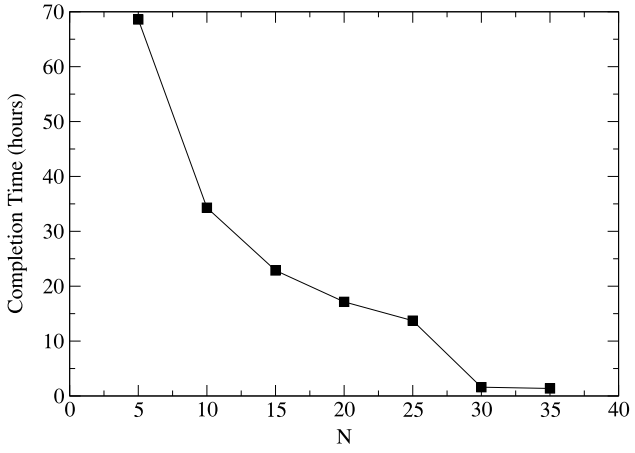

Fig. 10. Completion time for $J = 21, 22, E$ symmetry with ≈500,000,000 transitions for PH_3 against N MPI processes. At $N = 30$, I/O is effectively eliminated as all eigenvectors are stored in memory.

GAIN-MPI is configured to attach each MPI process to a node of GPUs. This can be done through the `NUM_GPUS` environment variable, `OMP_NUM_THREADS` environment variable and the mpirun processes per node options flag (i.e. `-perhost` or `-ppn` depending on the MPI library). The `NUM_GPUS` variable can be set to -1 for GAIN to detect all GPUs or >0 to use a specific number of GPUs. The *mem* keyword in the input file must be set to the total available to each MPI rank. An MPI process must have exclusive use of the GPUs and the GPUs cannot be shared between GAIN-MPI processes. Exclusive hosts, whereby each job is given an entire node, will have the -ppn flag set to 1, `OMP_NUM_THREADS` set to the total core count, `NUM_GPUS` set to -1 and *mem* set to the total memory of a single host. Slot based hosts, those that share nodes amongst other users, will have the `-ppn` flag set to 1, `OMP_NUM_THREADS` set to 1, `NUM_GPUS` set to 1 and *mem* set to the memory per core. Examples and a help file describing full usage are provided.

GAIN-MPI has been used to produce the SAlTY PH_3 [45], AYTY H_2CO [46] and hot SO_3 [48] linelists as efficiently as possible. Recently it has been used to complete the 1.4 billion transitions for the room temperature H_2O_2 linelist [58]. Here completing all transitions only took 5–6 h on a single GPU so an approach was used whereby multiple complete linelists with different parameters where prototyped and produced and the most suitable one selected for publication. This approach would be infeasible using the CPU version.

7.2. Extending GAIN-MPI

The `main.cpp` source handles the MPI management through classes. Included are number of abstract classes that can be inherited to interface with this code without dealing with the specific MPI communication. All classes inherit from a `BaseProcess` class that tracks MPI rank and process count. The classes that inherit from this are the `BasisSet` class providing the quantum numbers needed by GAIN. The abstract `States` class that handles the energies and filtering of states, the abstract `Eigenvector` class that contains functions to load eigenvectors across MPI processes and finally an `Output` class. Additionally there are the GPU based classes with `GpuManager` which exclusively deals with a single assigned GPU and the `MultiGpuManager` that manages multiple `GpuManager` classes and distributes work as well as interfaces with the main program. The supplementary material contains concrete versions of these classes specific to TROVE with the *TROVE* prefix (see Appendix A).

8. Conclusions

A new code has been implemented to compliment the TROVE software suite. The code accelerates the standard TROVE implementation by 10–1000 times with larger speedups favouring more difficult problems. This is accomplished by refactoring the half-linestrength code to exploit features of the basis set and utilizing all threads to perform maximal amount of work. A multi-GPU implementation allows for the multiple evaluation of transitions with minimal communication, this exhibits a strong scaling behaviour and allows for larger portions of the line list to be computed with fewer CPU hours and within wall-time limitations. This code, along

with improvements to diagonalization libraries and other algorithmic developments [59], will contribute to pushing the ExoMol Project towards bigger and more difficult molecules. Future work will look into implementing a version of this code on the Intel Xeon Phi and a version utilizing OpenCL instead of CUDA.

Acknowledgements

This work was supported by the European Research Council under Advanced Investigator Project 267219 and the COST action MOLIM (CM1405). The work presented here made use of the Emerald High Performance Computing facility provided via the Centre for Innovation (CfI). The CfI is formed from the universities of Bristol, Oxford, Southampton and UCL in partnership with STFC Rutherford Appleton Laboratory.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2017.01.013.

References

[1] P. Jensen, J. Mol. Spectrosc. 128 (1988) 478–501. http://dx.doi.org/10.1016/0022-2852(88)90164-6.
[2] D.W. Schwenke, J. Phys. Chem. 100 (1996) 2867–2884.
[3] J. Tennyson, M.A. Kostin, P. Barletta, G.J. Harris, O.L. Polyansky, J. Ramanlal, N.F. Zobov, Comput. Phys. Comm. 163 (2004) 85–116.
[4] I.N. Kozin, M.M. Law, J. Tennyson, J.M. Hutson, Comput. Phys. Comm. 163 (2004) 117–131.
[5] S.N. Yurchenko, W. Thiel, P. Jensen, J. Mol. Spectrosc. 245 (2007) 126–140. http://dx.doi.org/10.1016/j.jms.2007.07.009.
[6] A. Yachmenev, S.N. Yurchenko, J. Chem. Phys. 143 (2015) 014105. http://dx.doi.org/10.1063/1.4923039.
[7] L.A. Gribov, A.I. Pavlyuchko, Variational Methods for Solving Anharmonic Problems in the Theory of Vibrational Spectra of Molecules, Nauka, Moscow, 1998 (in Russian).
[8] A.I. Pavlyuchko, S.N. Yurchenko, J. Tennyson, Mol. Phys. 113 (2015) 1559–1575. http://dx.doi.org/10.1080/00268976.2014.992485.
[9] S. Carter, J.M. Bowman, J. Chem. Phys. 108 (1998) 4397–4404.
[10] S. Carter, J.M. Bowman, N.C. Handy, Theor. Chem. Acc. 100 (1998) 191–198. http://dx.doi.org/10.1007/s002140050379.
[11] S. Carter, A.R. Sharma, J.M. Bowman, P. Rosmus, R. Tarroni, J. Chem. Phys. 131 (2009) 224106. http://dx.doi.org/10.1063/1.3266577.
[12] J.M. Bowman, S. Carter, X.C. Huang, Intern. J. Quantum Chem. 22 (2003) 533–549. http://dx.doi.org/10.1080/0144235031000124163.
[13] N.C. Handy, S. Carter, Mol. Phys. 102 (2004) 2201–2205. http://dx.doi.org/10.1080/00268970410001728870.
[14] E. Mátyus, G. Czakó, A.G. Császár, J. Chem. Phys. 130 (2009) 134112. http://dx.doi.org/10.1063/1.3076742.
[15] M. Mladenovic, J. Chem. Phys. 112 (2000) 1070–1081.
[16] M. Mladenovic, Spectra Chimica Acta A 58 (2002) 795–807. http://dx.doi.org/10.1016/S1386-1425(01)00669-2.
[17] M. Mladenovic, Spectra Chimica Acta A 58 (2002) 809–824. http://dx.doi.org/10.1016/S1386-1425(01)00670-9.
[18] H. Wei, T. Carrington, J. Chem. Phys. 101 (1994) 1343–1360. http://dx.doi.org/10.1063/1.467827.
[19] P. Sarkar, N. Poulin, T. Carrington, J. Chem. Phys. 110 (1999) 10269–10274. http://dx.doi.org/10.1063/1.478960.
[20] J.C. Light, T. Carrington, Adv. Chem. Phys. 114 (2000) 263–310. http://dx.doi.org/10.1002/9780470141731.ch4.
[21] X.-G. Wang, T. Carrington Jr., J. Chem. Phys. 128 (2008) 194109. http://dx.doi.org/10.1063/1.2918498.
[22] X.-G. Wang, T. Carrington Jr., J. Chem. Phys. 130 (2009) 094101. http://dx.doi.org/10.1063/1.3077130.
[23] X.-G. Wang, T. Carrington Jr., J. Chem. Phys. 138 (2013) 104106. http://dx.doi.org/10.1063/1.4793474.
[24] G. Avila, T. Carrington Jr., J. Chem. Phys. 143 (2015) 214108. http://dx.doi.org/10.1063/1.4936294.
[25] T. Carrington Jr., Can. J. Phys. 93 (2015) 589–593. http://dx.doi.org/10.1139/cjc-2014-0590.
[26] D. Lauvergnat, A. Nauts, J. Chem. Phys. 116 (2002) 8560–8570. http://dx.doi.org/10.1063/1.1469019.
[27] H.G. Yu, J.T. Muckerman, J. Mol. Spectrosc. 214 (2002) 11–20. http://dx.doi.org/10.1006/jmsp.2002.8569.
[28] H. Guo, Rev. Comput. Chem. 25 (2007) 285–347. http://dx.doi.org/10.1002/9780470189078.ch7.
[29] F. Gatti, C. Iung, Phys. Rep. 484 (2009) 1–69. http://dx.doi.org/10.1016/j.physrep.2009.05.003.
[30] A.V. Nikitin, M. Rey, V.G. Tyuterev, J. Chem. Phys. 142 (2015) 094118. http://dx.doi.org/10.1063/1.4913520.
[31] G. Czakó, T. Furtenbacher, A.G. Császár, V. Szalay, Mol. Phys. 102 (2004) 2411–2423. http://dx.doi.org/10.1080/00268970420002749 91.
[32] J. Bloino, J. Phys. Chem. A 119 (2015) 5269–5287. http://dx.doi.org/10.1021/jp509985u.
[33] K. Sadri, D. Lauvergnat, F. Gatti, H.-D. Meyer, J. Chem. Phys. 141 (2014) 114101. http://dx.doi.org/10.1063/1.4895557.
[34] L. Lodi, J. Tennyson, O.L. Polyansky, J. Chem. Phys. 135 (2011) 034113.
[35] M. Grechko, O. Aseev, T.R. Rizzo, N.F. Zobov, L. Lodi, J. Tennyson, O.L. Polyansky, O.V. Boyarkin, J. Chem. Phys. 136 (2012) 244308.
[36] A. Petrignani, M. Berg, A. Wolf, I.I. Mizus, O.L. Polyansky, J. Tennyson, N.F. Zobov, M. Pavanello, L. Adamowicz, J. Chem. Phys. 141 (2014) 241104. http://dx.doi.org/10.1063/1.4904440.
[37] O.L. Polyansky, K. Bielska, M. Ghysels, L. Lodi, N.F. Zobov, J.T. Hodges, J. Tennyson, Phys. Rev. Lett. 114 (2015) 243001. http://dx.doi.org/10.1103/PhysRevLett.114.243001.
[38] E. Zak, J. Tennyson, O. L. Polyansky, L. Lodi, S.A. Tashkun, V.I. Perevalov, J. Quant. Spectrosc. Radiat. Transf. 177 (2016) 31–42. http://dx.doi.org/10.1016/j.jqsrt.2015.12.022.
[39] R. Warmbier, R. Schneider, A.R. Sharma, B.J. Braams, J.M. Bowman, P.H. Hauschildt, Astron. Astrophys. 495 (2009) 655–661. http://dx.doi.org/10.1051/0004-6361:200810983.
[40] S.N. Yurchenko, J. Tennyson, Mon. Not. R. Astron. Soc. 440 (2014) 1649–1661.
[41] M. Rey, A.V. Nikitin, V.G. Tyuterev, Astrophys. J. 789 (2014) 2. http://dx.doi.org/10.1088/0004-637X/789/1/2.
[42] S.N. Yurchenko, J. Tennyson, D.S. Amundsen, Astron. Astrophys.
[43] J. Tennyson, S.N. Yurchenko, Mon. Not. R. Astron. Soc. 425 (2012) 21–33. http://dx.doi.org/10.1111/j.1365-2966.2012.21440.x.
[44] S.N. Yurchenko, R.J. Barber, J. Tennyson, Mon. Not. R. Astron. Soc. 413 (2011) 1828–1834. http://dx.doi.org/10.1111/j.1365-2966.2011.18261.x.
[45] C. Sousa-Silva, A.F. Al-Refaie, J. Tennyson, S.N. Yurchenko, Mon. Not. R. Astron. Soc. 446 (2015) 2337–2347. http://dx.doi.org/10.1093/mnras/stu2246.
[46] A.F. Al-Refaie, S.N. Yurchenko, A. Yachmenev, J. Tennyson, Mon. Not. R. Astron. Soc. 448 (2015) 1704–1714. http://dx.doi.org/10.1093/mnras/stv091.
[47] A.F. Al-Refaie, O.L. Polyansky, R.I. Ovsyannikov, J. Tennyson, S.N. Yurchenko, Mon. Not. R. Astron. Soc. 461 (2016) 1012–1022. http://dx.doi.org/10.1093/mnras/stw1295.
[48] D.S. Underwood, J. Tennyson, S.N. Yurchenko, S. Clausen, A. Fateev, Mon. Not. R. Astron. Soc. 462 (2016) 4300–4313. http://dx.doi.org/10.1093/mnras/stw1828.
[49] S.N. Yurchenko, J. Tennyson, J. Bailey, M.D.J. Hollis, G. Tinetti, Proc. Natl. Acad. Sci. 111 (2014) 9379–9383. http://dx.doi.org/10.1073/pnas.1324219111.
[50] A.I. Pavlyuchko, S.N. Yurchenko, J. Tennyson, J. Chem. Phys. 142 (2015) 094309. http://dx.doi.org/10.1063/1.4913741.
[51] S.N. Yurchenko, W. Thiel, M. Carvajal, H. Lin, P. Jensen, Adv. Quant. Chem. 48 (2005) 209–238. http://dx.doi.org/10.1016/S0065-3276(05)48014-4.
[52] P.R. Bunker, P. Jensen, Molecular Symmetry and Spectroscopy, second ed., NRC Research Press, Ottawa, 1998.
[53] A.J. Stone, input.F90 a Fortran90 module for parsing text input, 2005, see http://www-stone.ch.cam.ac.uk/programs/.
[54] L.S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R.C. Whaley, ACM Trans. Math. Softw. 28 (2001) 135–151.
[55] nVidia, CUBLAS Library User Guide, nVidia, v5.0 edn., 2012. URL http://docs.nvidia.com/cublas/index.html.
[56] J. Tennyson, B.T. Sutcliffe, J. Chem. Phys. 77 (1982) 4061–4072. http://dx.doi.org/10.1063/1.444316.
[57] L.S. Rothman, I.E. Gordon, Y. Babikov, A. Barbe, D.C. Benner, P.F. Bernath, M. Birk, L. Bizzocchi, V. Boudon, L.R. Brown, A. Campargue, K. Chance, E.A. Cohen, L.H. Coudert, V.M. Devi, B.J. Drouin, A. Fayt, J.-M. Flaud, R.R. Gamache, J.J. Harrison, J.-M. Hartmann, C. Hill, J.T. Hodges, D. Jacquemart, A. Jolly, J. Lamouroux, R.J. Le Roy, G. Li, D.A. Long, O.M. Lyulin, C.J. Mackie, S.T. Massie, S. Mikhailenko, H.S.P. Müller, O.V. Naumenko, A.V. Nikitin, J. Orphal, V. Perevalov, A. Perrin, E.R. Polovtseva, C. Richard, M.A.H. Smith, E. Starikova, K. Sung, S. Tashkun, J. Tennyson, G.C. Toon, V.G. Tyuterev, G. Wagner, J. Quant. Spectrosc. Radiat. Transf. 130 (2013) 4–50. http://dx.doi.org/10.1016/jqsrt.2013.07.002.
[58] A.F. Al-Refaie, R.I. Ovsyannikov, O.L. Polyansky, S.N. Yurchenko, J. Tennyson, J. Mol. Spectrosc. 318 (2015) 84–90. http://dx.doi.org/10.1016/j.jms.2015.10.004.
[59] J. Tennyson, S.N. Yurchenko, Int. J. Quantum Chem. 117 (2017) 92–103. http://dx.doi.org/10.1002/qua.25190.