An Experimental OPENMP-Parallelized version of the FCclasses code version 2.21

April M. Van Winkle and John W. Silzel
School of Science, Technology and Health
Biola University Dept. of Chemistry
La Mirada, California, USA
john.silzel@biola.edu

January 21, 2017

0.1 Introduction

FCclasses[3] is a widely used time-independent code for the calculation of vibronic spectra in the harmonic approximation[6, 5, 7, 4]. For many systems of interest, a single-threaded implementation of FCclasses is very adequate. However, if finite temperatures (especially near ambient) are modeled, or if the system is a large one which includes many thermally accessible states, the execution time using a single thread may become inconveniently long. In these cases, a time-dependent treatment of the computational problem[1] is probably indicated. However, such code is not at this time (Fall 2016) widely available, and so the accompanying parallel-capable version of the time independent FCclasses code may be helpful to others as it has been in our work[2].

This document describes the modifications made in parallelizing, and especially some important limitations and memory issues that may be encountered in its use. As with any research code, this version does not include many checks and error traps that would normally be included in software intended for broad use by nonspecialists. The scope of this document is merely the parallelization of the FCclasses algorithm, and to document several known issues related to parallelization. For information concerning the calculations themselves, consult the FCClasses documentation supplied with the code.

This parallel version of FCclasses remains under the GNU license as specified in the accompanying original documentation.

0.2 Overview and Capability

The accompanying source code uses OPENMP directives to implement parallel execution regions in a thread-safe manner and is suitable for running FCclasses on a single node (a single computer system) having multiple processor cores. This version is not suitable for use on a multi-node computing cluster. Only portions of the code deemed most likely to benefit have been parallelized. In particular, only transitions involving 3 or more destination oscillators (Class 3 or greater in the terminology of the FCclasses algorithm) will run in parallel.

The code has been tested in a very rudimentary way only, and appears to give results identical to the test data supplied by FCclasses' author for both FC and HT calculations. However, this parallel version must be considered experimental and potentially unstable.

0.3 Compilation and Validation

No functional modification of the original FCclasses algorithm (version 2.1) has been made. Included with the FCclasses code is sample run output from FC and HT calculations made using the parallel version and the original sample data provided by FCclasses' authors. It is strongly suggested that workers experimenting with this version validate for themselves that their own compilations give suitable results (comparing their output to the original progam authors' output files) before relying on calculations made with this code.

This parallel version was compiled and tested using the gfortran(gcc) compiler, (versions 4.9.2 and 4.8.4) under Debian GNU/Linux 8 (jessie) 64-bit, both native and virtualized (VirtualBox 5.1.0). No other compilers have been tested at the time of this writing. The **-fopenmp** compiler switch must be specified to include the OPENMP libraries, and the **-o2** optimization level does not appear to "break" the code.

To compile the source, your command will probably resemble:

```
gfortran -fopenmp -o2 -o fcclasses221p.x fcclasses221p.f
```

No changes to the input or output files described in the FCclasses 2.1 manual are required.

0.4 Memory Usage and Management

The most significant change in this version is the handling of the large array FC that stores integrals in memory during execution. In the previous version, this two dimensional array of double precision was declared static. In this parallel version, the FC array is declared allocatable as a module variable. Note particularly that FC may be ALLOCATEd to be a size (mfc2 x mfc1) that may be larger than the available system RAM. This is possible because the ALLOCATE operation only allocates *address space* for the array, not RAM for the array itself. Actual reservation of RAM occurs implicitly as the array FC is initialized by the code. This is done based on the actual size requirement determined by the FCclasses algorithm for each class (C2 - C7). (Maximum dimensions of FC are given in the code by variables nfcmx1 and nfcmx2.)

This change was made because during parallel execution, a thread-private copy of FC must be available to each processor. The overall memory required by FC is thus the size of the array itself multiplied by the number of threads on the team. When a parallel section of the code is entered, threadprivate copies of FC are created for each thread, and these are released at the end of each parallel section, leaving only a single copy that contains the "core" integrals. The RAM required for parallel execution is therefore a function of the number of threads requested.

The modified FCclasses code includes module variables (see the *parvars* module) that monitor the required and actual sizes of the FC array during execution. This is done primarily so that some warning may be printed in the output file if a calculation is about to run out of memory. This can happen if the number of oscillators is large and simultaneously a large number of threads are to run in parallel. The modified code will reduce the number of threads in the event that RAM is running short, but of course cannot always prevent a crash if even a single copy of the FC array would exhaust system resources.

0.5 Running FCclasses in Parallel

This version of FCclasses accepts two optional command-line parameters. If used, these should be specified as integers. The first parameter specifies the *number of threads* that should be used, and the second specifies an approximate value for the number of gigabytes of RAM to be assumed present. For example, to run FCclasses in parallel using 8 cores and a maximum of 10 GB of RAM, the command is:

```
./fcclasses221p.x 8 10 < infile.inp > outfile.out &
```

If the parameters are omitted, then the code will use the default OPENMP number of threads (this is system dependent, but usually will be all the cores available on the system), and a memory size of 7 Gbytes.

If an immediate segmentation fault occurs on your system, it is probably because the default stack allocation is too small. Giving the command `ulimit -s 6000000` before execution will probably solve this.

0.5.1 Running FCclasses Using the Optional Bash Script

A bash script, **runfc**, is included with the code that may make it simpler to start parallel runs with FC-classes. Besides setting the stack size and passing command line parameters to FCclasses, the script also includes lines that set three OPENMP environment variables OMP_NUMTHREADS, OMP_STACKSIZE, and OMP_PROC_BIND. See the OPENMP documentation (widely available online) for the use of these and other environment variables. At the end of the run the script also renames the FCclasses output files (fort.18, fort.21, etc.) so that the output files are easily related to the input file used.

To use the bash script, one might give the command

```
./runfc myrun 8 6 &
```

to force the use of 8 cores and 6 Gb RAM, or to use the default number of threads and RAM:

```
./runfc myrun &
```

In each cases, the system will seek the input file myrun.inp (note that the ".inp" is not typed on the command line) and subsequent to the run generate output files myrun.out, myrun.fort.18, myrun.fort.21, etc. See the end of this document for a listing of the bash script. Of course, you may need to set execution permission on the bash script when first placing it on your system.

0.6 Hints for Efficient Use

- **Not all sections of the FCclasses code run in parallel.** In particular, parallel execution is only implemented for calculation of Franck-Condon factors for classes 3 and higher. Calculation of classes 1 and 2 is typically quite efficient and need not be parallelized. There will be periods of execution where only one core is utilized, this is not necessarily a failure to run in parallel.
- **Execution speed will not always scale with the thread count.** The FCclasses code involves a large amount of memory access, which will show up on most UNIX systems as "sys" usage rather than "user" (e.g. in the **top** utility.) If a large number of threads are being used, memory access may become the "rate limiting step". When more CPU is going to "sys" than "user", running fewer threads may give faster execution overall.
- **Monitoring progress.** In this version, the log output has been modified to provide information during parallel execution that is displayed in the following format (e.g.):

  ```
  M: 2/4 C4 INDX: 105/109 THD: 6/8 nInt/THD:n1 FCT/THD:n2
  ```

  this line indicates that FCclasses is working on mother state 2 (of 4 mother states total), evaluating Class 4 transitions, that the outer loop index is 105 out of 109, and that this outer loop index has been assigned to thread 6 (of 8). For reasons of avoiding needless variable reductions, the number of integrals computed is displayed on a per-thread basis (n1), and the total FC factors likewise (n2). At the end of the parallel section of code, the overall FC and integral totals are displayed for all threads.

0.7 Bash Script Listing:

```
#!/bin/bash
echo "FCclasses 2.21 Parallel Version"
JOB=$1
nThds=$2:-0
GbRAM=$3:-7
export OMP_STACKSIZE=6000000
export OMP_PROC_BIND="TRUE"
export OMP_NUMTHREADS=1
ulimit -s 6000000
echo "Running your job:  " $JOB
./fcclasses221p.x $nThds $GbRAM < $JOB.inp > $JOB.out &
wait
mv fort.18 $JOB.fort.18
mv fort.21 $JOB.fort.21
mv fort.8 $JOB.fort.8
mv fort.9 $JOB.fort.9
```

Bibliography

[1] Francisco Jos Avila Ferrer, Javier Cerezo, Juan Soto, Roberto Improta, and Fabrizio Santoro. First-principle computation of absorption and fluorescence spectra in solution accounting for vibronic structure, temperature effects and solvent inhomogenous broadening. *Computational and Theoretical Chemistry*, 1040-1041:328337, Jul 2014.

[2] Katrina Tao Hua Lin and John W. Silzel. Relation of molecular structure to franckcondon bands in the visible-light absorption spectra of symmetric cationic cyanine dyes. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, 142(0):210 – 219, 2015.

[3] Fabrizio Santoro. Fcclasses, a fortran 77 code., May 2012.

[4] Fabrizio Santoro and Vincenzo Barone. Computational approach to the study of the lineshape of absorption and electronic circular dichroism spectra. *International Journal of Quantum Chemistry*, 110(2):476–486, 2010.

[5] Fabrizio Santoro, Roberto Improta, Alessandro Lami, Julien Bloino, and Vincenzo Barone. Effective method to compute franck-condon integrals for optical spectra of large molecules in solution. *Journal of Chemical Physics*, 126(8):084509–, February 2007. FCClasses.

[6] Fabrizio Santoro, Alessandro Lami, Roberto Improta, and Vincenzo Barone. Effective method to compute vibrationally resolved optical spectra of large molecules at finite temperature in the gas phase and in solution. *Journal of Chemical Physics*, 126(18):184102–, May 2007. FCClasses Major paper.

[7] Fabrizio Santoro, Alessandro Lami, Roberto Improta, Julien Bloino, and Vincenzo Barone. Effective method for the computation of optical spectra of large molecules at finite temperature including the duschinsky and herzbergteller effect: The qx band of porphyrin as a case study. *The Journal of Chemical Physics*, 128(22):–, 2008.