PMP Molscat: a parallel version of Molscat v. 14

George C. McBane

Grand Valley State University

1 Overview

Molscat is a popular program for computation of inelastic scattering cross sections, written by S. Green and J. Hutson. PMP Molscat provides a parallelized version of the program. It can be used with Beowulf-type cluster systems, grid systems such as Apple's Xgrid, symmetric multiprocessor systems such as SGI Altix or IBM p690, or on collections of independent serial machines. On clusters or SMP machines it uses MPI message passing.

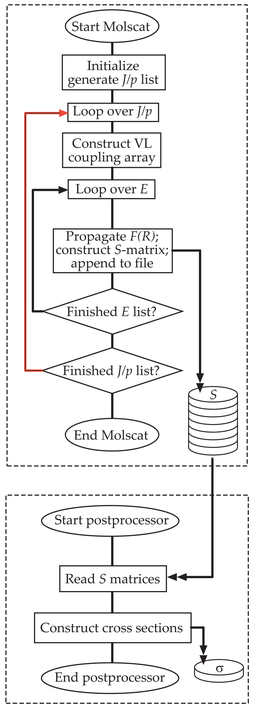
2 Basic structure of a quantum scattering calculation

Computation of scattering cross sections with the close coupled (CC) method requires a series of "propagations". Propagations are carried out for many values (tens to hundreds) of the total angular momentum J, two values of the parity $p = \pm 1$, and all total energies E of interest. Each propagation yields an S matrix. Once all the S-matrices are known, their elements can be combined to determine integral cross sections, differential cross sections, pressure broadening cross sections, and so on. The computation time for the whole calculation is dominated by the time required for the propagations.

In a "coupled states" (CS) calculation, the structure of the calculation is the same, but p is replaced by j_z, the projection of the rotor angular momentum on the intermolecular axis. Typically 1–50 values of j_z are required, but the individual propagations take much less time than in the CC method, and all propagations take about the same amount of time.

Molscat places the E loop inside the J/p loop. This structure avoids unnecessary recomputation of the angular momentum coupling array VL, which is independent of E. The program writes all the S-matrices to a single file, one after another. After the Molscat run, a second "post-processor" program reads this S-matrix file and computes cross sections.

The structure of a calculation with serial Molscat is shown at right. The strategy of PMP Molscat is to parallelize the loop over angular momentum and parity, shown in red in the diagram.



3 PMP Molscat program design

3.1 Philosophy

PMP stands for "poor man's parallel". The package was originally conceived as a means of dividing up the work of a large cross section calculation among several independent serial machines, without requiring any direct interprocess communication. The main design criterion is therefore *No large data arrays are passed between processes.*

This design decision produces two important limitations:

- PMP Molscat does not parallelize the loop over total energy, because that would require passing the VL array between processes.
- PMP Molscat requires the use of postprocessor programs even for calculation of integral cross sections; it does not compute any cross sections as part of the main run, because that would require passing S matrices between processes.

In return for these limitations, PMP Molscat is simple in structure, runs on a wide variety of parallel machines, and does not require high network speed.

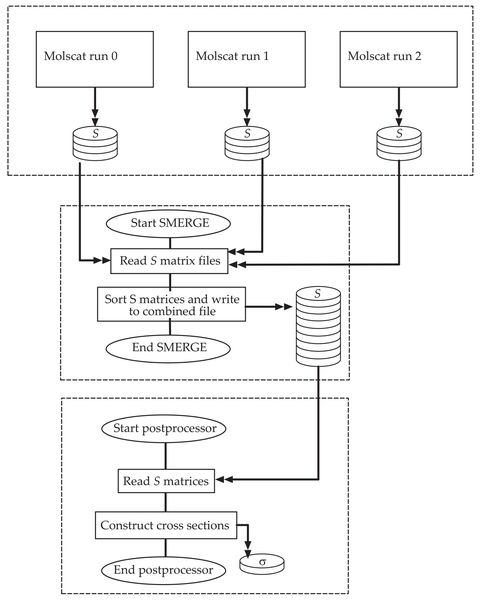
3.2 Components

PMP Molscat provides two basic tools:

1. A utility program, smerge, that can read in S-matrix files generated by several different Molscat runs, sort them, and write them out as a single file that looks like it was generated by one long run of serial Molscat.
2. A version of the main Molscat program with additional code for parallel computers or computer clusters. The extra code uses MPI message passing. It spawns several Molscat processes, and divides up the J/p pairs from a single Molscat input file among the processes. Each process writes out its own S-matrix file.

3.3 Use of smerge

smerge combines S-matrix files from several different Molscat runs into a single, properly sorted file that can be read by the postprocessor programs.



4 True PMP: Using smerge without MPI

Users without MPI but with access to groups of serial machines can use smerge to combine results from separate runs of serial Molscat. The user must prepare a set of Molscat input files, each of which specifies a subset of the J/p pairs needed. Each of those files can then be distributed to a different machine for calculations with serial Molscat. Batch-job schedulers such as PBS or "grid computing" systems such as Xgrid, if they are available, simplify the task of distributing the input files and collecting the resulting output files.

For example, if three machines are available, the three different input files might contain the lines

machine 1: `JTOTL = 0, JTOTU=30, JSTEP=3`

machine 2: `JTOTL = 1, JTOTU=30, JSTEP=3`

machine 3: `JTOTL = 2, JTOTU=30, JSTEP=3`

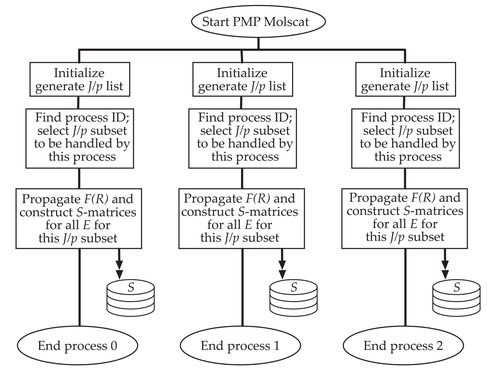
Machine 1 would then do calculations for $J = 0, 3, 6\ldots$, machine 2 would do $J = 1, 4, 7\ldots$, and so on. Taken together, these three runs would include all values of J with $J \leq 30$.

When all the runs are finished, their S-matrix files can then be combined by smerge and cross sections generated by the postprocessor programs as usual.

5 Using MPI: static and dynamic task allocation

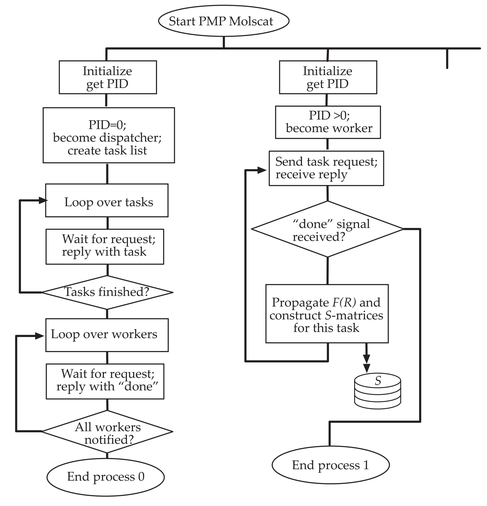
For users with machines running MPI ("Beowulf" clusters or SMP machines), PMP Molscat provides two different mechanisms for allocating all the J/p pairs ("tasks") from a single Molscat input file among the available processes.

5.1 Static allocation



Each process uses a simple algorithm to select the J/p pairs it is responsible for and carries out the scattering calculations for those pairs. The static allocation mechanism is especially effective when all the tasks take the same amount of time, as in CS calculations or CC calculations when $J > j_{\max}$,where $j_{\max}$ is the highest rotational quantum number included in the basis set. Under these conditions the user should specify a number of processes (and CPUs) that is a divisor of the number of J/p pairs. Parallel performance is then excellent, even up to one CPU per task.

5.2 Dynamic allocation



One process acts as "dispatcher", assigning tasks to the "worker" processes until all tasks have been assigned. Tasks are handed out in decreasing order of length. The dynamic allocation mechanism works well when there is a large number of tasks of varying lengths. It is most often used for CC calculations. With some operating systems it is efficient to "overload" one CPU with both the dispatcher and one computational process, so it is not necessary to dedicate an entire CPU to the dispatcher's job. In typical CC calculations, numbers of CPUs up to about a third the number of total J/p pairs give good efficiency.

6 Performance

6.1 Scaling for CS calculations

This table shows results for a small static-allocation job with 14 total tasks, taking a total of about 5 seconds. Notice the efficiency increases obtained when the number of CPUs is a divisor of the number of tasks. The "total time" increases with number of CPUs because of the duplicated initialization work; this effect is less important for larger jobs.

CPUs	total time/s	max process time/s	speedup	efficiency
2	4.18	2.18	1.9	96%
3	4.15	1.48	2.8	93
4	4.29	1.27	3.4	84
5	4.29	0.93	4.6	92
6	4.34	0.95	4.6	76
7	4.45	0.65	6.8	98
8	4.36	0.66	6.6	83
14	4.60	0.35	13.1	94

6.2 Scaling for CC calculations

This table shows scaling performance for a small CC calculation. This calculation had 48 tasks ($0 \leq J \leq 23$, $p = \pm 1$), and took about 1000 s total. Note the effect of "overloading" one CPU with both the dispatcher and a computational process.

Model	CPUs	MPI proc.	speedup	efficiency
static	4	4	3.4	84%
	5	5	4.5	90
	7	7	6.0	85
	8	8	6.2	77
	16	16	10.3	65
dynamic	8	8	6.9	87
	8	9	7.8	93
	16	17	14.4	90

6.3 Performance in production calculations

Performance for several real calculations is shown below. The number of MPI processes is listed; for the calculation marked *, each process was an 8-way SMP node dedicated to a single propagation using multithreaded BLAS. All these calculations were run on the DataStar machine at the San Diego Supercomputer Center.

Calc type	Model	Tasks	$N_{\max}$	MPI proc.	speedup	efficiency
CS	static	6105	99	48	47	>95%
CS	static	2109	97	304	240	79
CC	static	22	4376	22*	18.3	83
CC	dynamic	82	756	24	23.0	96

7 Portability and availability

The smerge program is written in near-standard Fortran 77, using a few extensions that are widely available in modern compilers (including g77). It should compile easily on any machine that can compile Molscat.

The package is available through a link on the main Molscat web page at `http://www.giss.nasa.gov/tools/molscat/`, or directly at `http://faculty.gvsu.edu/mcbaneg/pmpmolscat/index.html`. If you use it, please send me email at `mcbaneg@gvsu.edu` so I can keep you informed of updates to the program.

8 Acknowledgements

Thanks to Jeremy Hutson for advice, to Ian Bush for his original PVM-based parallel version, to Christian Trefftz for parallel programming advice, to Teck-Ghee Lee and Brian Stewart for bug reports, and to several computer centers (OSC in Columbus, GWDG in Göttingen, SDSC in San Diego, and GVSU) for access to parallel machines.