Appendix A

Technical Details

A.1 Mixed or Double precision

GROMACS can be compiled in either mixed or double precision. Documentation of previous GROMACS versions referred to "single precision", but the implementation has made selective use of double precision for many years. Using single precision for all variables would lead to a significant reduction in accuracy. Although in "mixed precision" all state vectors, i.e. particle coordinates, velocities and forces, are stored in single precision, critical variables are double precision. A typical example of the latter is the virial, which is a sum over all forces in the system, which have varying signs. In addition, in many parts of the code we managed to avoid double precision for arithmetic, by paying attention to summation order or reorganization of mathematical expressions. The default configuration uses mixed precision, but it is easy to turn on double precision by adding the option `-DGMX_DOUBLE=on` to `cmake`. Double precision will be 20 to 100% slower than mixed precision depending on the architecture you are running on. Double precision will use somewhat more memory and run input, energy and full-precision trajectory files will be almost twice as large.

The energies in mixed precision are accurate up to the last decimal, the last one or two decimals of the forces are non-significant. The virial is less accurate than the forces, since the virial is only one order of magnitude larger than the size of each element in the sum over all atoms (sec. B.1). In most cases this is not really a problem, since the fluctuations in the virial can be two orders of magnitude larger than the average. Using cut-offs for the Coulomb interactions cause large errors in the energies, forces, and virial. Even when using a reaction-field or lattice sum method, the errors are larger than, or comparable to, the errors due to the partial use of single precision. Since MD is chaotic, trajectories with very similar starting conditions will diverge rapidly, the divergence is faster in mixed precision than in double precision.

For most simulations, mixed precision is accurate enough. In some cases double precision is required to get reasonable results:

- normal mode analysis, for the conjugate gradient or l-bfgs minimization and the calculation and diagonalization of the Hessian

- long-term energy conservation, especially for large systems

A.2 Environment Variables

GROMACS programs may be influenced by the use of environment variables. First of all, the variables set in the `GMXRC` file are essential for running and compiling GROMACS. Some other useful environment variables are listed in the following sections. Most environment variables function by being set in your shell to any non-NULL value. Specific requirements are described below if other values need to be set. You should consult the documentation for your shell for instructions on how to set environment variables in the current shell, or in config files for future shells. Note that requirements for exporting environment variables to jobs run under batch control systems vary and you should consult your local documentation for details.

Output Control

1. `GMX_CONSTRAINTVIR`: print constraint virial and force virial energy terms.
2. `GMX_MAXBACKUP`: GROMACS automatically backs up old copies of files when trying to write a new file of the same name, and this variable controls the maximum number of backups that will be made, default 99. If set to 0 it fails to run if any output file already exists. And if set to -1 it overwrites any output file without making a backup.
3. `GMX_NO_QUOTES`: if this is explicitly set, no cool quotes will be printed at the end of a program.
4. `GMX_SUPPRESS_DUMP`: prevent dumping of step files during (for example) blowing up during failure of constraint algorithms.
5. `GMX_TPI_DUMP`: dump all configurations to a `.pdb` file that have an interaction energy less than the value set in this environment variable.
6. `GMX_VIEW_XPM`: `GMX_VIEW_XVG`, `GMX_VIEW_EPS` and `GMX_VIEW_PDB`, commands used to automatically view `.xvg`, `.xpm`, `.eps` and `.pdb` file types, respectively; they default to `xv`, `xmgrace`, `ghostview` and `rasmol`. Set to empty to disable automatic viewing of a particular file type. The command will be forked off and run in the background at the same priority as the GROMACS tool (which might not be what you want). Be careful not to use a command which blocks the terminal (*e.g.* `vi`), since multiple instances might be run.
7. `GMX_VIRIAL_TEMPERATURE`: print virial temperature energy term
8. `GMX_LOG_BUFFER`: the size of the buffer for file I/O. When set to 0, all file I/O will be unbuffered and therefore very slow. This can be handy for debugging purposes, because it ensures that all files are always totally up-to-date.
9. `GMX_LOGO_COLOR`: set display color for logo in `ngmx`.
10. `GMX_PRINT_LONGFORMAT`: use long float format when printing decimal values.

11. GMX_COMPELDUMP: Applies for computational electrophysiology setups only (see section 6.6). The initial structure gets dumped to .pdb file, which allows to check whether multimeric channels have the correct PBC representation.

Debugging

1. GMX_PRINT_DEBUG_LINES: when set, print debugging info on line numbers.
2. GMX_DD_NST_DUMP: number of steps that elapse between dumping the current DD to a PDB file (default 0). This only takes effect during domain decomposition, so it should typically be 0 (never), 1 (every DD phase) or a multiple of nstlist.
3. GMX_DD_NST_DUMP_GRID: number of steps that elapse between dumping the current DD grid to a PDB file (default 0). This only takes effect during domain decomposition, so it should typically be 0 (never), 1 (every DD phase) or a multiple of nstlist.
4. GMX_DD_DEBUG: general debugging trigger for every domain decomposition (default 0, meaning off). Currently only checks global-local atom index mapping for consistency.
5. GMX_DD_NPULSE: over-ride the number of DD pulses used (default 0, meaning no override). Normally 1 or 2.

Performance and Run Control

1. GMX_DO_GALACTIC_DYNAMICS: planetary simulations are made possible (just for fun) by setting this environment variable, which allows setting epsilon_r = -1 in the .mdp file. Normally, epsilon_r must be greater than zero to prevent a fatal error. See www.gromacs.org for example input files for a planetary simulation.
2. GMX_ALLOW_CPT_MISMATCH: when set, runs will not exit if the ensemble set in the .tpr file does not match that of the .cpt file.
3. GMX_CUDA_NB_EWALD_TWINCUT: force the use of twin-range cutoff kernel even if rvdw = rcoulomb after PP-PME load balancing. The switch to twin-range kernels is automated, so this variable should be used only for benchmarking.
4. GMX_CUDA_NB_ANA_EWALD: force the use of analytical Ewald kernels. Should be used only for benchmarking.
5. GMX_CUDA_NB_TAB_EWALD: force the use of tabulated Ewald kernels. Should be used only for benchmarking.
6. GMX_CUDA_STREAMSYNC: force the use of cudaStreamSynchronize on ECC-enabled GPUs, which leads to performance loss due to a known CUDA driver bug present in API v5.0 NVIDIA drivers (pre-30x.xx). Cannot be set simultaneously with GMX_NO_CUDA_STREAMSYNC.
7. GMX_CYCLE_ALL: times all code during runs. Incompatible with threads.
8. GMX_CYCLE_BARRIER: calls MPI_Barrier before each cycle start/stop call.

9. GMX_DD_ORDER_ZYX: build domain decomposition cells in the order (z, y, x) rather than the default (x, y, z).

10. GMX_DD_USE_SENDRECV2: during constraint and vsite communication, use a pair of MPI_SendRecv calls instead of two simultaneous non-blocking calls (default 0, meaning off). Might be faster on some MPI implementations.

11. GMX_DLB_BASED_ON_FLOPS: do domain-decomposition dynamic load balancing based on flop count rather than measured time elapsed (default 0, meaning off). This makes the load balancing reproducible, which can be useful for debugging purposes. A value of 1 uses the flops; a value ¿ 1 adds (value - 1)*5% of noise to the flops to increase the imbalance and the scaling.

12. GMX_DLB_MAX_BOX_SCALING: maximum percentage box scaling permitted per domain-decomposition load-balancing step (default 10)

13. GMX_DD_RECORD_LOAD: record DD load statistics for reporting at end of the run (default 1, meaning on)

14. GMX_DD_NST_SORT_CHARGE_GROUPS: number of steps that elapse between re-sorting of the charge groups (default 1). This only takes effect during domain decomposition, so should typically be 0 (never), 1 (to mean at every domain decomposition), or a multiple of nstlist.

15. GMX_DETAILED_PERF_STATS: when set, print slightly more detailed performance information to the .log file. The resulting output is the way performance summary is reported in versions 4.5.x and thus may be useful for anyone using scripts to parse .log files or standard output.

16. GMX_DISABLE_SIMD_KERNELS: disables architecture-specific SIMD-optimized (SSE2, SSE4.1, AVX, etc.) non-bonded kernels thus forcing the use of plain C kernels.

17. GMX_DISABLE_CUDA_TIMING: timing of asynchronously executed GPU operations can have a non-negligible overhead with short step times. Disabling timing can improve performance in these cases.

18. GMX_DISABLE_GPU_DETECTION: when set, disables GPU detection even if mdrun was compiled with GPU support.

19. GMX_DISABLE_PINHT: disable pinning of consecutive threads to physical cores when using Intel hyperthreading. Controlled with mdrun -nopinht and thus this environment variable will likely be removed.

20. GMX_DISRE_ENSEMBLE_SIZE: the number of systems for distance restraint ensemble averaging. Takes an integer value.

21. GMX_EMULATE_GPU: emulate GPU runs by using algorithmically equivalent CPU reference code instead of GPU-accelerated functions. As the CPU code is slow, it is intended to be used only for debugging purposes. The behavior is automatically triggered if non-bonded calculations are turned off using GMX_NO_NONBONDED case in which the non-bonded calculations will not be called, but the CPU-GPU transfer will also be skipped.

22. GMX_ENX_NO_FATAL: disable exiting upon encountering a corrupted frame in an .edr file, allowing the use of all frames up until the corruption.

23. GMX_FORCE_UPDATE: update forces when invoking mdrun -rerun.

24. GMX_GPU_ID: set in the same way as the mdrun option -gpu_id, GMX_GPU_ID allows the user to specify different GPU id-s, which can be useful for selecting different devices on different compute nodes in a cluster. Cannot be used in conjunction with -gpu_id.

25. GMX_IGNORE_FSYNC_FAILURE_ENV: allow mdrun to continue even if a file is missing.

26. GMX_LJCOMB_TOL: when set to a floating-point value, overrides the default tolerance of 1e-5 for force-field floating-point parameters.

27. GMX_MAX_MPI_THREADS: sets the maximum number of MPI-threads that mdrun can use.

28. GMX_MAXCONSTRWARN: if set to -1, mdrun will not exit if it produces too many LINCS warnings.

29. GMX_NB_GENERIC: use the generic C kernel. Should be set if using the group-based cutoff scheme and also sets GMX_NO_SOLV_OPT to be true, thus disabling solvent optimizations as well.

30. GMX_NB_MIN_CI: neighbor list balancing parameter used when running on GPU. Sets the target minimum number pair-lists in order to improve multi-processor load-balance for better performance with small simulation systems. Must be set to a positive integer, the default value is optimized for NVIDIA Fermi and Kepler GPUs, therefore changing it is not necessary for normal usage, but it can be useful on future architectures.

31. GMX_NBLISTCG: use neighbor list and kernels based on charge groups.

32. GMX_NBNXN_CYCLE: when set, print detailed neighbor search cycle counting.

33. GMX_NBNXN_EWALD_ANALYTICAL: force the use of analytical Ewald non-bonded kernels, mutually exclusive of GMX_NBNXN_EWALD_TABLE.

34. GMX_NBNXN_EWALD_TABLE: force the use of tabulated Ewald non-bonded kernels, mutually exclusive of GMX_NBNXN_EWALD_ANALYTICAL.

35. GMX_NBNXN_SIMD_2XNN: force the use of 2x(N+N) SIMD CPU non-bonded kernels, mutually exclusive of GMX_NBNXN_SIMD_4XN.

36. GMX_NBNXN_SIMD_4XN: force the use of 4xN SIMD CPU non-bonded kernels, mutually exclusive of GMX_NBNXN_SIMD_2XNN.

37. GMX_NO_ALLVSALL: disables optimized all-vs-all kernels.

38. GMX_NO_CART_REORDER: used in initializing domain decomposition communicators. Rank reordering is default, but can be switched off with this environment variable.

39. GMX_NO_CUDA_STREAMSYNC: the opposite of GMX_CUDA_STREAMSYNC. Disables the use of the standard cudaStreamSynchronize-based GPU waiting to improve performance when using CUDA driver API ealier than v5.0 with ECC-enabled GPUs.

40. GMX_NO_INT, GMX_NO_TERM, GMX_NO_USR1: disable signal handlers for SIGINT, SIGTERM, and SIGUSR1, respectively.

41. GMX_NO_NODECOMM: do not use separate inter- and intra-node communicators.

42. GMX_NO_NONBONDED: skip non-bonded calculations; can be used to estimate the possible performance gain from adding a GPU accelerator to the current hardware setup – assuming that this is fast enough to complete the non-bonded calculations while the CPU does bonded force and PME computation.

43. GMX_NO_PULLVIR: when set, do not add virial contribution to COM pull forces.

44. GMX_NOCHARGEGROUPS: disables multi-atom charge groups, *i.e.* each atom in all non-solvent molecules is assigned its own charge group.

45. GMX_NOPREDICT: shell positions are not predicted.

46. GMX_NO_SOLV_OPT: turns off solvent optimizations; automatic if GMX_NB_GENERIC is enabled.

47. GMX_NSCELL_NCG: the ideal number of charge groups per neighbor searching grid cell is hard-coded to a value of 10. Setting this environment variable to any other integer value overrides this hard-coded value.

48. GMX_PME_NTHREADS: set the number of OpenMP or PME threads (overrides the number guessed by mdrun.

49. GMX_PME_P3M: use P3M-optimized influence function instead of smooth PME B-spline interpolation.

50. GMX_PME_THREAD_DIVISION: PME thread division in the format "x y z" for all three dimensions. The sum of the threads in each dimension must equal the total number of PME threads (set in GMX_PME_NTHREADS).

51. GMX_PMEONEDD: if the number of domain decomposition cells is set to 1 for both x and y, decompose PME in one dimension.

52. GMX_REQUIRE_SHELL_INIT: require that shell positions are initiated.

53. GMX_REQUIRE_TABLES: require the use of tabulated Coulombic and van der Waals interactions.

54. GMX_SCSIGMA_MIN: the minimum value for soft-core σ. **Note** that this value is set using the sc-sigma keyword in the .mdp file, but this environment variable can be used to reproduce pre-4.5 behavior with respect to this parameter.

55. GMX_TPIC_MASSES: should contain multiple masses used for test particle insertion into a cavity. The center of mass of the last atoms is used for insertion into the cavity.

56. GMX_USE_GRAPH: use graph for bonded interactions.

57. GMX_VERLET_BUFFER_RES: resolution of buffer size in Verlet cutoff scheme. The default value is 0.001, but can be overridden with this environment variable.

58. GMX_VERLET_SCHEME: convert from group-based to Verlet cutoff scheme, even if the cutoff_scheme is not set to use Verlet in the .mdp file. It is unnecessary since the -testverlet option of mdrun has the same functionality, but it is maintained for backwards compatibility.

59. MPIRUN: the mpirun command used by g_tune_pme.

60. MDRUN: the mdrun command used by g_tune_pme.

61. GMX_NSTLIST: sets the default value for nstlist, preventing it from being tuned during mdrun startup when using the Verlet cutoff scheme.

62. GMX_USE_TREEREDUCE: use tree reduction for nbnxn force reduction. Potentially faster for large number of OpenMP threads (if memory locality is important).

Analysis and Core Functions

1. GMX_QM_ACCURACY: accuracy in Gaussian L510 (MC-SCF) component program.

2. GMX_QM_ORCA_BASENAME: prefix of .tpr files, used in Orca calculations for input and output file names.

3. GMX_QM_CPMCSCF: when set to a nonzero value, Gaussian QM calculations will iteratively solve the CP-MCSCF equations.

4. GMX_QM_MODIFIED_LINKS_DIR: location of modified links in Gaussian.

5. DSSP: used by do_dssp to point to the dssp executable (not just its path).

6. GMX_QM_GAUSS_DIR: directory where Gaussian is installed.

7. GMX_QM_GAUSS_EXE: name of the Gaussian executable.

8. GMX_DIPOLE_SPACING: spacing used by g_dipoles.

9. GMX_MAXRESRENUM: sets the maximum number of residues to be renumbered by grompp. A value of -1 indicates all residues should be renumbered.

10. GMX_FFRTP_TER_RENAME: Some force fields (like AMBER) use specific names for N- and C- terminal residues (NXXX and CXXX) as .rtp entries that are normally renamed. Setting this environment variable disables this renaming.

11. GMX_PATH_GZIP: gunzip executable, used by g_wham.

12. GMX_FONT: name of X11 font used by ngmx.

13. GMXTIMEUNIT: the time unit used in output files, can be anything in fs, ps, ns, us, ms, s, m or h.
14. GMX_QM_GAUSSIAN_MEMORY: memory used for Gaussian QM calculation.
15. MULTIPROT: name of the multiprot executable, used by the contributed program do_multiprot.
16. NCPUS: number of CPUs to be used for Gaussian QM calculation
17. GMX_ORCA_PATH: directory where Orca is installed.
18. GMX_QM_SA_STEP: simulated annealing step size for Gaussian QM calculation.
19. GMX_QM_GROUND_STATE: defines state for Gaussian surface hopping calculation.
20. GMX_TOTAL: name of the total executable used by the contributed do_shift program.
21. GMX_ENER_VERBOSE: make g_energy and eneconv loud and noisy.
22. VMD_PLUGIN_PATH: where to find VMD plug-ins. Needed to be able to read file formats recognized only by a VMD plug-in.
23. VMDDIR: base path of VMD installation.
24. GMX_USE_XMGR: sets viewer to xmgr (deprecated) instead of xmgrace.

A.3 Running GROMACS in parallel

By default GROMACS will be compiled with the built-in thread-MPI library. This library handles communication between threads on a single node more efficiently than using an external MPI library. To run GROMACS in parallel over multiple nodes, e.g. on a cluster, you need to configure and compile GROMACS with an external MPI library. All supercomputers are shipped with MPI libraries optimized for that particular platform, and there are several good free MPI implementations; OpenMPI is usually a good choice. Note that MPI and thread-MPI support are mutually incompatible.

In addition to MPI parallelization, GROMACS supports also thread-parallelization through OpenMP. MPI and OpenMP parallelization can be combined, which results in, so called, hybrid parallelization. It can offer better performance and scaling in some cases.

See www.gromacs.org for details on the use and performance of the different parallelization schemes.

A.4 Running GROMACS on GPUs

As of version 4.6, GROMACS has native GPU support through CUDA. Note that GROMACS only off-loads the most compute intensive parts to the GPU, currently the non-bonded interactions, and does all other parts of the MD calculation on the CPU. The requirements for the CUDA code are

an Nvidia GPU with compute capability ≥ 2.0, i.e. at least Fermi class. In many cases `cmake` can auto-detect GPUs and the support will be configured automatically. To be sure GPU support is configured, pass the `-DGMX_GPU=on` option to `cmake`. The actual use of GPUs is decided at run time by `mdrun`, depending on the availability of (suitable) GPUs and on the run input settings. A binary compiled with GPU support can also run CPU only simulations. Use `mdrun -nb cpu` to force a simulation to run on CPUs only. Only simulations with the Verlet cut-off scheme will run on a GPU. To test performance of old tpr files with GPUs, you can use the `-testverlet` option of `mdrun`, but as this doesn't do the full parameter consistency check of `grommp`, you should not use this option for production simulations. Getting good performance with GROMACS on GPUs is easy, but getting best performance can be difficult. Please check www.gromacs.org for up to date information on GPU usage.