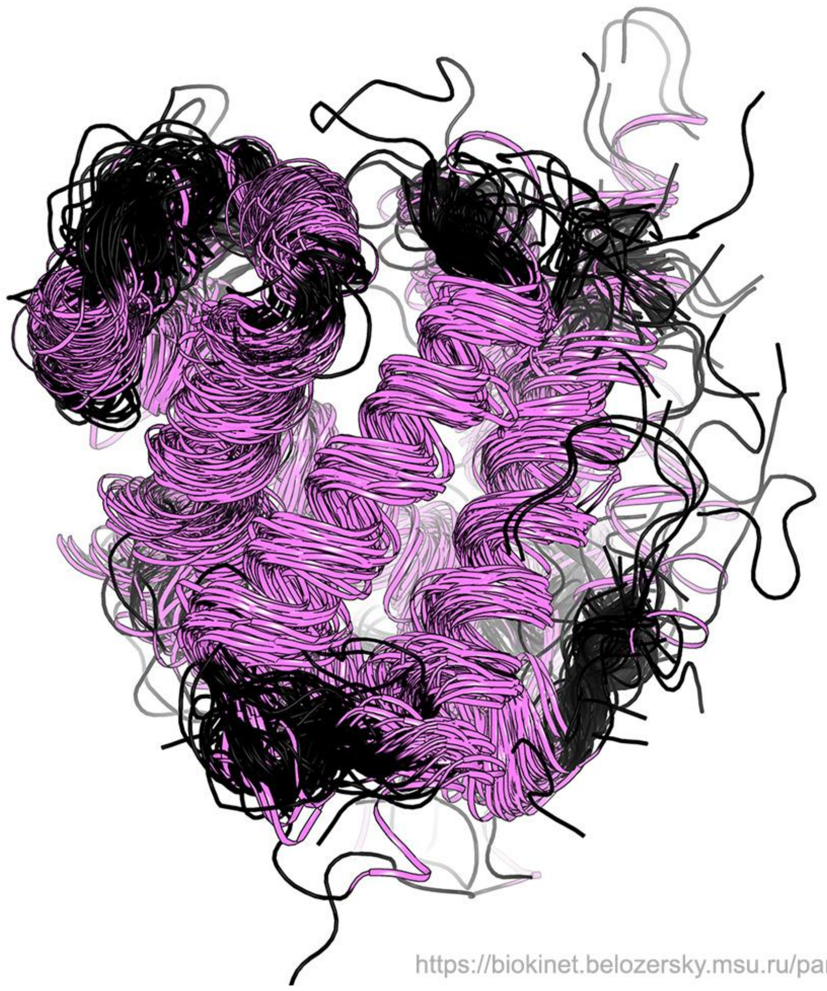
parMATT

Parallel multiple alignment of protein 3D-structures with translations and twists for distributed-memory systems

The User's Manual



parMATT v.1.2

November 5th, 2020

Contents

Abstract

A 3D-alignment of multiple protein structures is fundamentally important for a variety of tasks in modern biology and becomes more time-consuming with the increase of the number of PDB records to be compared. Ten years ago it was common to superimpose just a few protein structures due to a limited amount of 3D-data deposited at that time. Today, non-redundant collections of protein superfamilies are represented by hundreds of 3D-records, making it problematic to use the available single-CPU software to perform such a superimposition. More than 146 million sequence entries of the currently known proteins are deposited in the UniProtKB database, and as the PDB database demonstrates a geometric growth we are facing further increase in the number of known protein structures corresponding to diverse superfamilies, ruling out the use of single-CPU 3D-alignment programs at a daily routine in the future.

The parMATT is a hybrid MPI/pthreads/OpenMP parallel re-implementation of the MATT algorithm designed to benefit from the growing availability of structural data by accelerating multiple structural alignment at large-scale analysis of protein families/superfamilies. The parMATT can be faster than MATT on a single multi-core CPU, and provides a much greater speed-up on distributed-memory systems, i.e., computing clusters and supercomputers hosting memory-independent computing nodes. The parMATT can significantly accelerate the time-consuming process of building a multiple structural alignment from a large collection of 3D-models of homologous proteins. The output of MATT and parMATT are identical.

The parMATT is the first and only program currently available which supports the MPI level of parallelism at aligning multiple protein structures. Its source code is distributed under the GNU public license version 2.0 and available for download at https://biokinet.belozersky.msu.ru/parMATT.

Version history

Version	Date	What's new
1.0	2018-Feb-20	-
1.1	2019-Apr-14	(1) Load balancing on a large number of nodes was improved. (2) The "*Partial Alignments*" step – i.e., the post processing of the finally created structural superimposition – was parallelized using openMP to run on all cores of one multi-core CPU. The "*Partial Alignments*" step is not part of the MATT algorithm and has no effect on the structural superimposition of input proteins. This procedure is enabled by default in the MATT's source code, according to the original documentation, to make visual inspection of the sequence representation of the structural alignment easier; however, can have a very significant performance impact, particularly on alignments of large numbers of structures. In parMATT the "*Partial Alignments*" step is disabled by default. To activate this step add the '*-p1*' flag to the command line string. See this page to understand the *"Partial alignments"*.
1.2	2020-Nov-05	Fixed a bug that could lead to an incorrect format of the output PDB file with 3D-alignment if the superimposed structures contained more than 99999 atoms. In previous versions of the program, in such cases, atomids after 100000 shifted the data line one character to the left, which could cause incompatibility with subsequent programs. As of this release, the atomid counter is reset to 1 after 99999, which is the usual workaround for this well-known limitation of the classic PDB format.

Introduction

A hybrid parallel programming technique has been used to re-implement the MATT algorithm and produce a faster program – parMATT, whose unique feature is the ability to accelerate the multiple protein structural alignment by running on multiple nodes of multiprocessor computer systems. The most computationally demanding steps of the algorithm – the initial construction of pairwise alignments between all input structures and further iterative extension of the multiple alignment – were parallelized using the MPI and pthreads, and the concluding refinement step was accelerated on one CPU by introducing the OpenMP support (i.e., the shared-memory multithreading). The temporary data are stored in the RAM and distributed across all allocated nodes to efficiently use the available resources. See the parMATT's publication for details.

Installation of parMATT from sources is straightforward, does not require significant investment of time from the user, and can be performed by free tools (i.e., GNU C++ and MPI compilers). The software does not have a graphical interface and has to be executed from a command-line. The input to parMATT is a set of protein structures in the PDB format, and the primary output is (1) a file in the PDB format with 3D superimposition of all input structures, and (2) a file in the FASTA format with a corresponding structure-based sequence alignment of the common structural core (i.e., structural equivalences which are shared by all proteins from the input set). The parMATT inherits the bioinformatics part (i.e., the algorithm), the input and the output formats, options and environmental variables from the MATT source code, and thus the parMATT's output files are identical to that of the MATT. The parMATT can be launched on a regular desktop multi-core CPU, but its key advantage is the ability to run on distributed-memory multiprocessor systems, i.e., computing clusters and supercomputers hosting memory-independent computing nodes.

This User's Manual focuses on the multiprocessor-related features as well as changes to the input options and environmental variables which have been introduced in parMATT. Implementation of parMATT in the laboratory practice as well as guidelines for collecting large data sets of 3D-models of homologous proteins will be also discussed.

For a full list of options and features of the "Multiple alignment with translations and twists" algorithm please refer to the MATT's user manual and the web-page http://matt.cs.tufts.edu/.

Prerequisites

The parMATT is not hardware-specific and is expected to run on any architecture under a Linux/Unix operating system. The software was primarily developed for multiprocessor computer systems with distributed memory. It implements MPI for communication between processors (nodes) and pthreads to utilize multiple cores within each processor (node). Therefore, to achieve the best performance with parMATT you should use a computing cluster or a supercomputer hosting *memory-independent computing nodes*, i.e. powerful computers with multiple CPUs. You can also use parMATT to construct a multiple structural alignment on a single desktop CPU with multiple computing *cores* (i.e., *shared-memory computing system*).

To compile the parMATT binary from the source code you need a Linux/Unix computer system with MPI environment and pthreads support. This means that you would normally need an MPI compiler (e.g. Intel MPI or openMPI) and a C compiler (e.g., Intel *icc* or GNU *gcc*). You can install the free GNU tools openMPI (https://www.open-mpi.org/) and GCC (https://gcc.gnu.org) – both packages should be available from a developers software repository on any Linux distribution (e.g., in openSuSE you can install both packages in YaST). Or you can go for the all-in-one Intel Parallel Composer XE/Intel Parallel Studio XE pack which is commercial, though non-commercial license terms are available in particular cases.

Compilation

After the **prerequisites** have been met enter the project folder and run:

```
cd parMATT

make
```

You should see a similar command line output:

```
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/matt/AssemblyOrder.o src/matt/AssemblyOrder.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/RMSD.o
src/matt/RMSD.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/Score.o
src/matt/Score.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/matt/MultipleAlignment.o src/matt/MultipleAlignment.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/pdb.o
src/matt/pdb.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/matt/MultipleAlignmentOutput.o src/matt/MultipleAlignmentOutput.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/Matt.o
src/matt/Matt.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/Extend.o
src/matt/Extend.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/chain.o
src/matt/chain.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/Vector.o
src/matt/Vector.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/util.o
src/matt/util.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/OctTree.o
src/matt/OctTree.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/FileReader.o
src/matt/FileReader.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/secondary.o
src/matt/secondary.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/matt/Protein.o
src/matt/Protein.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/mpi/mpi_pairscores.o src/mpi/mpi_pairscores.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/mpi/mpi_serializer.o src/mpi/mpi_serializer.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/mpi/mpi_helpers.o
src/mpi/mpi_helpers.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o
src/mpi/mpi_multiple_alignment.o src/mpi/mpi_multiple_alignment.c
mpicc -pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3   -c -o src/mpi/mpi_pool.o
src/mpi/mpi_pool.c
mpicc -o bin/parMatt src/matt/AssemblyOrder.o src/matt/RMSD.o src/matt/Score.o
src/matt/MultipleAlignment.o src/matt/pdb.o src/matt/MultipleAlignmentOutput.o src/matt/Matt.o
src/matt/Extend.o src/matt/chain.o src/matt/Vector.o src/matt/util.o src/matt/OctTree.o
src/matt/FileReader.o src/matt/secondary.o src/matt/Protein.o src/mpi/mpi_pairscores.o
src/mpi/mpi_serializer.o src/mpi/mpi_helpers.o src/mpi/mpi_multiple_alignment.o src/mpi/mpi_pool.o -
pthread -ggdb -Ithird_party -Iinclude -std=gnu99 -fopenmp -O3 -lm
```

If the compilation has been successful the executable binary file `parMatt` would be placed in the `'./bin'` folder.

The parMATT's options and variables

The parMATT supports the same list of options (i.e., command line arguments) as the original MATT program.

Special characteristics of parMATT are:

- The '`-d`' option for "display status" is not supported;
- The '`-p`' option for "partial alignments" is disabled by default (i.e., `-p0`). See this page to understand the *"Partial alignments"*;
- The '`-t`' option specifies the number of pthreads ***t*** to be spawned *on each node* to process subtasks (i.e., build the alignment). Additional pthreads will spawned for administrative purposes - one pthread for communication on each slave node, and two pthreads on the master node – one for communication and one for tasks' scattering. E.g. if you run parMATT with '`-t` ***4***' parameter than 4+1 pthreads would be spawned on each slave node (4 for the alignments and 1 for communication) and 4+2 pthreads would be spawned on the master node (4 for the alignments, 1 for communication, and 1 for tasks' scattering). The default is '`-t` 1' and would significantly downgrade the performance in most cases as it assumes that only a single computing core is available in a CPU. You should set this parameter manually for a specific hardware – see recommendations and examples below.

Recommendations on selecting the '`-t` ***t***' number of pthreads: ***t*** should be set equal to the number of physical cores on your nodes. If your PC/cluster has hardware hyperthreading enabled, you may set the ***t*** to the number of threads (usually equal to the number of cores × 2). E.g., to run parMATT on multiple nodes, where each node hosts an Intel Core i7 CPU with ***4*** physical cores (8 logical threads), you should use '`-t` ***4***'. Or to run parMATT on multiple nodes, where each node hosts an Intel Xeon CPU E5-2697 v3 CPU (***14*** physical cores or 28 threads), you should use '`-t` ***14***'.

parMATT supports the same list of environment variables as the original MATT program (i.e., `MATT_SEARCH_PATH`, `MATT_PDB_PATH`, `MATT_PARAMS`). To set a variable use the `export` feature of the shell:

```
export VAR="VALUE"
```

Then you can check if the assignment worked:

```
echo $VAR
```

A new environment variable `MATT_LOG_LVL` is introduced to control the amount of data to be displayed in the standard output of parMATT. This variable is intended for developers and is not needed for the 'scientific' use. This variable does not influence the results or the format of the output alignments produced by the parMATT.

The possible values are:

- `MATT_LOG_LVL 0` – log only timings for pairwise alignments and iterative part of the algorithm;
- `MATT_LOG_LVL 1` – all above plus the information about tasks sent;
- `MATT_LOG_LVL 2` – all above plus timings for each alignment and barrier wait time (default).

To set the variable use the `export` feature of the shell:

```
export MATT_LOG_LVL="0"
```

Then you can check if the assignment worked:

```
echo $MATT_LOG_LVL
```

The parMATT's input

The input to parMATT is a set of protein structures in the PDB format. **One PDB file should represent a *single protein chain*.**

There are two ways of providing the list of input PDB files to parMATT/MATT. The first one is via a *list* file – plain text file stating the paths to each input PDB file on a separate line, one below the other:

```
sbatch -N 8 ompi parMatt -L input.list -t 14 -o output
```

The second way is to provide each input PDB file path as a separate command line argument:

```
sbatch -N 8 ompi parMatt file1.pdb file2.pdb ... -t 14 -o output
sbatch -N 8 ompi parMatt *.pdb -t 14 -o output
sbatch -N 8 ompi parMatt `find ./ -name *.pdb` -t 14 -o output
```

The second way may appear more convenient, but it is impractical. The parMATT was designed to accommodate computationally hard tasks, e.g. to align large data sets of protein structures. Large datasets = a lot of input PDB file = long command line arguments. In other words, the paths to a large number of PDB files will most likely exceed the default maximum length of command line arguments allowed by the Linux/Unix shell. Therefore, we recommend the *list*-file based input when using the parMATT. You can prepare the *list* file using the command:

```
find /absolute/path/to/pdbs -name "*.pdb" > list.file
```

Please note that the command above will produce a *list*-file with absolute paths to your input PDB files, i.e., this *list*-file can be used to execute parMATT from any location on your local filesystem.

Running parMATT

The parMATT software is faster than MATT on a single desktop CPU and can provide much greater acceleration on distributed-memory systems, i.e., computing clusters and supercomputers hosting memory-independent computing nodes. The difference between running MATT on a local computer and running parMATT on a computing cluster/supercomputer is explained below:

1. to run on multiple nodes (i.e., multiple CPUs) parMATT has to be launched as an MPI program by the appropriate MPI utility (not required for local execution on a desktop computer);
2. the '-t ***t***' parameter should be set equal to the number of physical cores in the CPUs which are used in your computing cluster/supercomputer, i.e., the '-t ***t***' parameter sets the numbers of cores to be used on *each* node, and the number of nodes should be set as a separate parameter to the MPI utility (see an example below). If your PC/cluster has hardware hyperthreading enabled, you may set the ***t*** to the number of threads (usually equal to the number of cores × 2).

This may sound complicated, but it's not. You just need to add a few MPI-related "words" as a prefix to the regular command, and you need to learn the number of physical cores in the CPUs used in your computing cluster/supercomputer – from your administrator or by exploring the vendor's site (the CPU model can be found in the `/proc/cpuinfo` file). Once you know the command and the number of physical cores in your CPU model, running parMATT will be as easy as running any other program on your local computer.

To run the parMATT you need to execute the '`bin/parMatt`' binary in the MPI environment and provide the '-t ***t***' parameter. The exact command will depend on the particular setup of your computing system. A few examples are provided below.

Launch parMATT **<u>locally</u>** on 4 physical cores of a single Desktop CPU:

```
/path/to/parMatt -t 4 -L input.list -o output
```

Launch parMATT on 8 nodes (i.e., 8 CPUs), 14 physical cores on each node, using the *mpirun*:

```
mpirun -np 8 /path/to/parMatt -t 14 -L input.list -o output
```

Launch parMATT compiled with OpenMPI on 8 nodes (i.e., 8 CPUs), 14 physical cores on each node, using the Slurm Workload Manager (*sbatch* and *ompi* script):

```
sbatch -N 8 ompi /path/to/parMatt -t 14 -L input.list -o output
```

Launch parMATT compiled with Intel MPI on 8 nodes (i.e., 8 CPUs), 14 physical cores on each node, using the Slurm Workload Manager (*sbatch* and *impi* script):

```
sbatch -N 8 impi /path/to/parMatt -t 14 -L input.list -o output
```

Launch parMATT on 8 nodes (i.e., 8 CPUs), 14 physical cores on each node, using the Cleo Workload Manager:

```
cleo-submit -np 8 /path/to/parMatt -t 14 -L input.list -o output
```

If you do not know how to launch a program on multiple nodes (i.e., CPUs) of your computing cluster or supercomputer you should contact the administrator.

The parMATT's output

The parMATT inherits the bioinformatics part (i.e., the algorithm), the input and the output formats, options and environmental variables from the MATT source code, and thus **the parMATT's output alignment is identical to that of the MATT**.

The primary parMATT's output is

- a file in the **PDB format with 3D superimposition of all input structures**;
- a file in the **FASTA format** with a corresponding structure-based sequence alignment**.** By default**,** the FASTA file **contains only the common structural core alignment** (i.e., structural equivalences which are shared by all proteins from the input set), and can be further enriched by implementing the *"Partial alignments"* algorithm. See this page to understand the *"Partial alignments"*.

The following particular files are produced by parMATT/MATT on successful completion:

- the **coordinate representation of a multiple structural alignment**, i.e., a PDB file with aligned coordinates of all 3D-models from the input;
- the **sequence representation of a multiple structural alignment**, i.e., a sequence alignment file in FASTA format;
- a **text file with a summary** of the input PDBs (the pairwise comparison tree) and the output superimposition (number of residues in the core alignment, RMSD of the core alignment, the MATT raw score and the sequence representation of the alignment in the PHYLIP format);
- a **Rasmol script** to highlight aligned residues.

parMATT's output: understanding *"Partial alignments"*

As explained above, the output of parMATT is the 3D-alignment in the PDB format and its sequence representation in the FASTA format. The conversion of the 3D-alignment (i.e., superimposition of atom coordinates in the 3D-space) into the alphabet form is actually not as trivial as it may seem. In fact, it is complex and highly resource-demanding process.

By default, the parMATT will produce the FASTA alignment of only the common core residues, i.e. structural equivalences which are shared by all proteins from the input set, as shown below on Figure A. This representation may be further improved by implementing the "*Partial Alignments*" algorithm. Formally, the "*Partial Alignments*" step is not part of the MATT algorithm and has no effect on the structural superimposition of input proteins (i.e., the output PDB files containing the 3D-alignment will be identical with or without this step). When this step is enabled, the sub-alignments of residues that are not equivalent in ***all*** proteins but can be matched in ***some*** homologs (i.e., the partial alignments) will be included in the FASTA file to accompany the common core alignment, as shown below on Figure B. This will produce a more content-rich output and facilitate its visual inspection, but will require additional computer time.

To activate the "*Partial Alignments*" step add the '***–p1***' flag (no space) to the parMATT command. The default is '***–p0***'.

```
>prot1  ---GRST----HSLHY-EDDL------
>prot2  -------GS--HSLRY-R---DG----
>prot3  ---------GSHSLRY-W-----DG--
>prot4  -----------HSLRY-Q-------DG
>prot5  ADS--------HSLRCGP---------
common core        ***** *
```

A. Sequence representation of 3D-alignment with *"Partial alignments"* disabled

```
>prot1  -GRSTHSLHY-EDDL
>prot2  --GS-HSLRY-RDG-
>prot3  --GS-HSLRY-WDG-
>prot4  -----HSLRY-QDG-
>prot5  A-DS-HSLRCGP---
common core  ***** *
```

B. Sequence representation of 3D-alignment with *"Partial alignments"* enabled

The parMATT's parallel performance and scalability

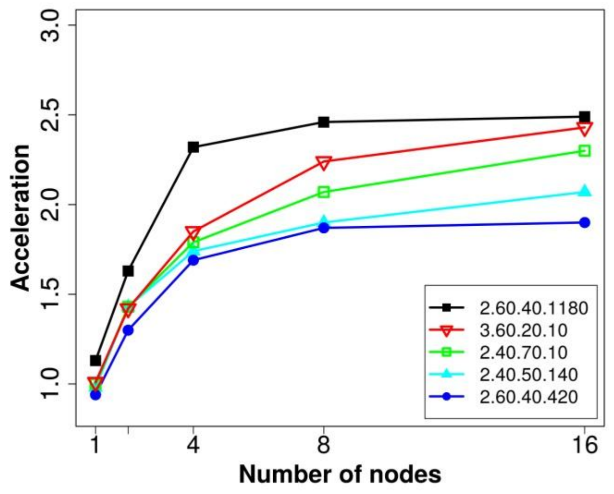
The parMATT was tested on 1-256 nodes (i.e., 14 cores per node, up to 3584 cores in total). The results are discussed in detail in the corresponding parMATT publication (https://dx.doi.org/10.1093/bioinformatics/btz224).

In brief, the observed computational efficiency and saturation point where performance growth stopped were proportional to the number of PDB structures in the input set and their size (i.e., the number of amino acid residues), with a better performance on a larger number of nodes observed when aligning more populated sets of a bigger protein structures.

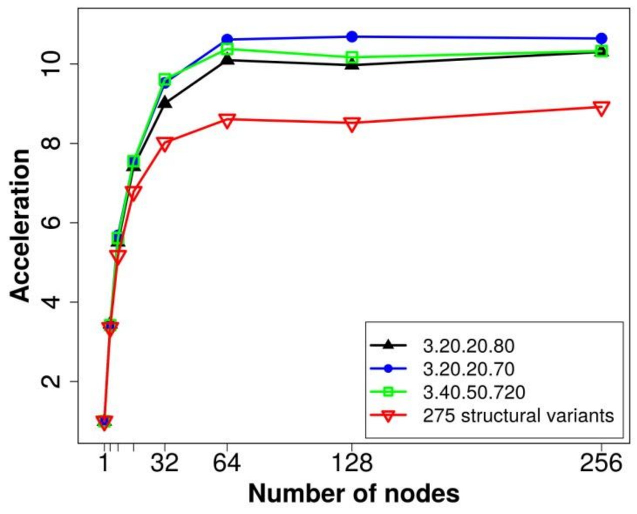
Thirty non-redundant sets of hundreds of protein structures sharing a common structural core have been constructed based on the CATH classification of superfamilies and used to evaluate computational performance of the parMATT software. Parallel performance of the alignment of smaller sets which contained 43-93 protein structures with an average size of 90-228 amino acid residues stagnated starting from 4-16 nodes with a speed-up of at most x2.49 (Fig. 1, A). On the contrast, the speed-up reached at most x10.62 and stagnated starting from 64 nodes when superimposing 275-341 protein structures with an average size of 191-348 amino acid residues (Fig. 1, B and Fig. 2, A).

To further evaluate the potentials of parMATT at aligning more massive datasets two series of structural variants of a 351 amino acids long protein were produced by molecular dynamics: (1) a test set of 2000 conformations and (2) a control set of 275 conformations which was similar to the CATH superfamily set '3.20.20.80' in terms of the number of structures and their size. Both sets were submitted to MATT and parMATT for a 3D-alignment. The 3D-superimposition of 2000 structural variants by parMATT showed a maximum speed-up of x31.7 on 128 nodes compared to the performance of MATT on a single multi-core node, i.e., 4.08 hours when running on a distributed-memory system compared to 5.39 days on one 14-core CPU (Fig. 1, C and Fig. 2, B). The performance of parMATT on the control set of 275 conformations was equivalent to that on the CATH superfamily set '3.20.20.80' (Fig. 1, B and Fig. 2, A). Thus, the observed improvement of performance on a larger number of nodes was due to the increase of the number of input structures and was not a consequence of aligning structural variants which corresponded to the same protein sequence, as parMATT was used to perform a 3D-superimposition.

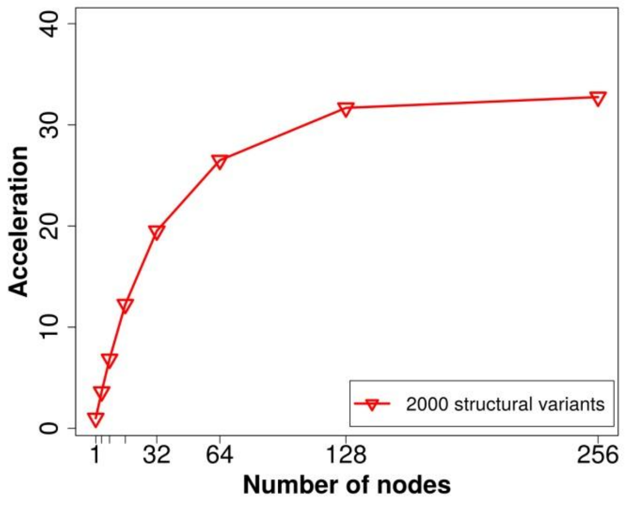
These results indicate that parMATT can significantly accelerate superimposition of hundreds of structures of different proteins by running on 2-64 nodes. Utilization of a larger number of nodes can be practically useful when aligning more populated collections of protein structures.



(A) Smaller CATH sets

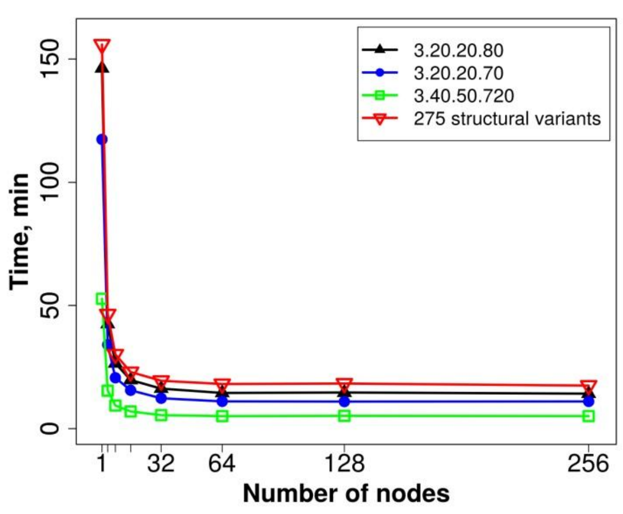


(B) Larger CATH sets

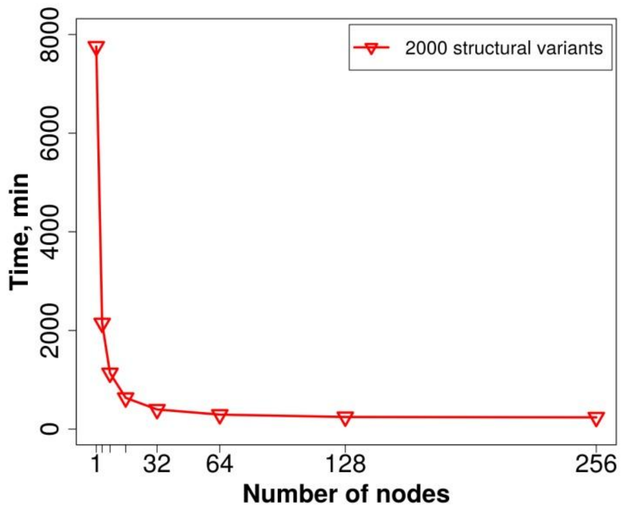


(C) MD-generated ensemble

Fig. 1. Acceleration provided by parMATT at constructing multiple 3D-alignments of protein structures when running on distributed-memory system. The speed-up achieved by parMATT on 1-256 nodes was calculated based on the performance of MATT in the openMP mode on all cores of one node. Each node contained 14 computing cores. (A) Parallel performance of 3D-alignment of smaller CATH superfamily sets which contained 43-93 protein structures with an average size of 90-228 amino acid residues. The corresponding running times were within several minutes. (B) Parallel performance of 3D-alignment of larger CATH superfamily sets which contained 275-341 protein structures with an average size of 191-348 amino acid residues, and a control set of 275 structural variants of a 351 amino acids long protein produced by the molecular dynamics which was similar to the CATH superfamily set '3.20.20.80' in terms of the number of structures and their size. The corresponding running times are provided in Fig. 2, A. (C) Parallel performance of 3D-alignment of a set of 2000 structural variants of a 351 amino acids long protein produced by the molecular dynamics. The corresponding running times are provided in Fig. 2, B



(A) Larger CATH sets



(B) MD-generated ensemble

Fig. 2. The running time of parMATT at constructing multiple 3D-alignments of protein structures in a parallel environment of a distributed-memory system. Each node contained 14 computing cores. (A) Duration of 3D-alignment of larger CATH superfamily sets which contained 275-341 protein structures with an average size of 191--348 amino acid residues, and a control set of 275 structural variants of a 351 amino acids long protein produced by the molecular dynamics which was similar to the CATH superfamily set '3.20.20.80' in terms of the number of structures and their size. The corresponding acceleration plots are shown in Fig. 1, B. (B) Duration of 3D-alignment of a set of 2000 structural variants of a 351 amino acids long protein produced by the molecular dynamics. The corresponding acceleration plots are shown in Fig. 1, C.

Post processing of the parMATT/MATT's 3D alignment file

On successful completion the parMATT/MATT produces, among other output, a file in the PDB format with 3D superimposition of all input structures. In this large single PDB file all input proteins are listed as individual chains and this format may not always be convenient for further analysis, especially when comparing large datasets of structures. In practice, **you may want to split this large single PDB alignment file into multiple PDB files each corresponding to an individual protein**, but preserving the common coordinate space (i.e., so that these multiple PDB files appear aligned if opened altogether, e.g., by a 3D graphical viewer like PyMol).

To post process the parMATT/MATT's 3D alignment file you can use an accessory script `splitMATT2chains.sh`, which can be downloaded from the parMATT's web-page. Please note that THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND. PLEASE CHECK MANUALLY THE CONSISTENCY OF THE GENERATED OUTPUT. An example is discussed below. To reproduce this example on your own computer you can download the input files by using this [link]. The command line syntax is intended from Linux Bash shell.

Extract the files from archive and enter the project folder:

```
tar xzf Example_split.tar.gz
cd Example_split
```

Launch the script:

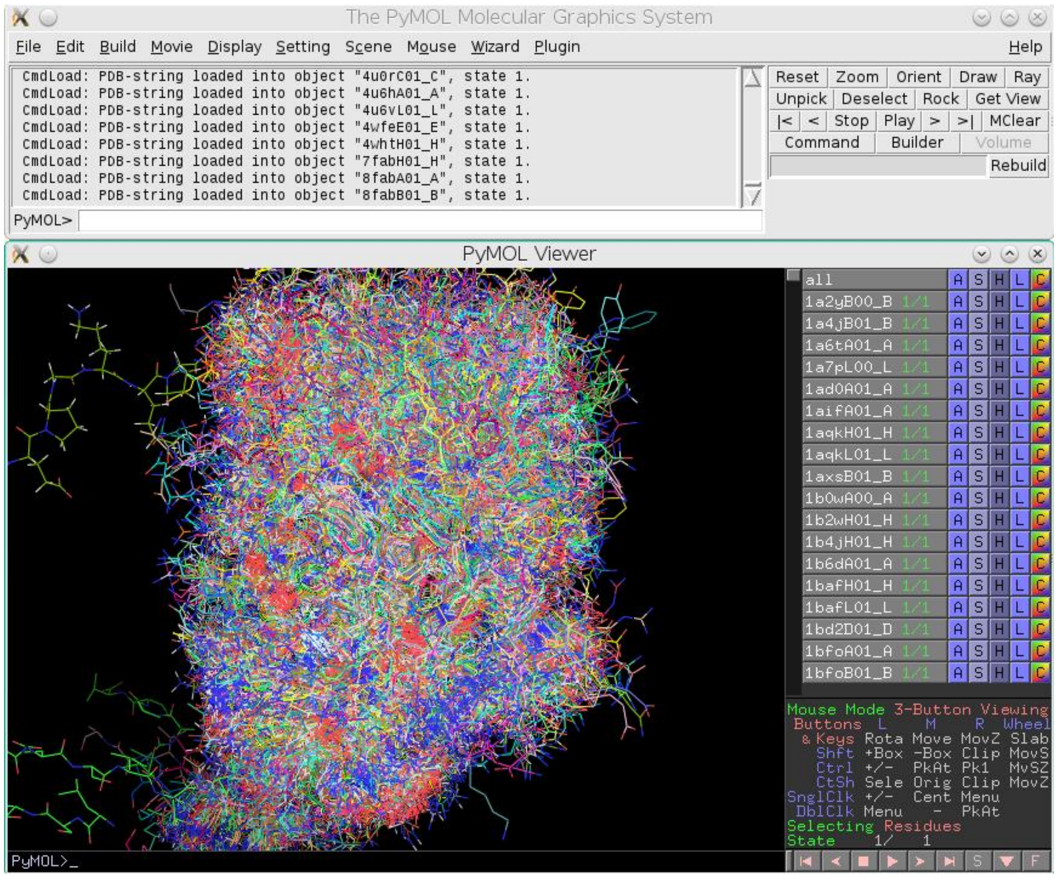
```
/path/to/splitMATT2chains.sh 2.60.40.10.pdb 2.60.40.10.txt output
```

This command will write 643 PDB files corresponding to the aligned proteins to folder `output`. Each file will be entitled as the original PDB files which were submitted to parMATT/MATT as input (in this example the files will be named according to their PDB codes).

You can now enter the `output` folder and load all 643 PDB files into the PyMol graphical viewer (this operation requires a modern Desktop computer with a significant amount of RAM):

```
cd output
pymol *pdb
```

All structures, loaded from separate PDB files, will appear superimposed in the 3D viewer:


The PyMOL Molecular Graphics System
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
Reset Zoom Orient Draw Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Command Builder Volume
Rebuild
PyMOL Viewer

```
CmdLoad: PDB-string loaded into object "4u0rC01_C", state 1.
CmdLoad: PDB-string loaded into object "4u6hA01_A", state 1.
CmdLoad: PDB-string loaded into object "4u6vL01_L", state 1.
CmdLoad: PDB-string loaded into object "4wfeE01_E", state 1.
CmdLoad: PDB-string loaded into object "4whtH01_H", state 1.
CmdLoad: PDB-string loaded into object "7fabH01_H", state 1.
CmdLoad: PDB-string loaded into object "8fabA01_A", state 1.
CmdLoad: PDB-string loaded into object "8fabB01_B", state 1.
PyMOL>
```

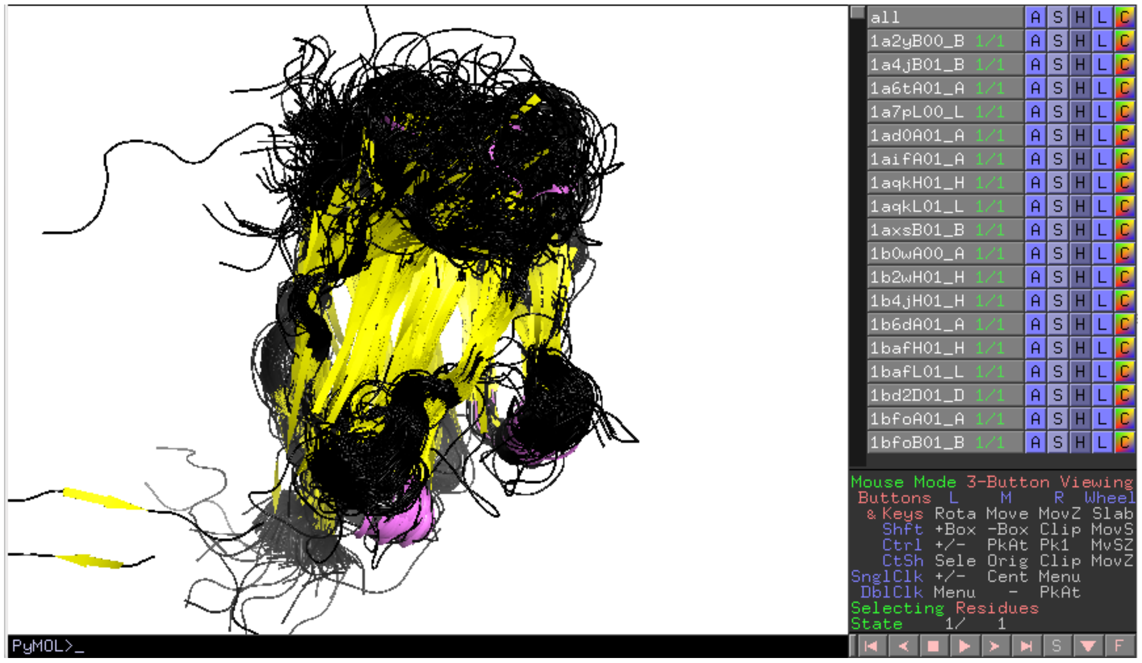
```
all
1a2yB00_B 1/1
1a4jB01_B 1/1
1a6tA01_A 1/1
1a7pL00_L 1/1
1ad0A01_A 1/1
1aifA01_A 1/1
1aqkH01_H 1/1
1aqkL01_L 1/1
1axsB01_B 1/1
1b0wA00_A 1/1
1b2wH01_H 1/1
1b4jH01_H 1/1
1b6dA01_A 1/1
1bafH01_H 1/1
1bafL01_L 1/1
1bd2D01_D 1/1
1bfoA01_A 1/1
1bfoB01_B 1/1
```

```
Mouse Mode 3-Button Viewing
 Buttons  L    M    R  Wheel
  & Keys Rota Move MovZ Slab
    Shft +Box -Box Clip MovS
    Ctrl +/-  PkAt Pk1  MvSZ
    CtSh Sele Orig Clip MovZ
SnglClk +/-  Cent Menu
 DblClk Menu  -   PkAt
Selecting Residues
State     1/    1
PyMOL>_
```

You may want to apply additional settings to create a nicer view. Execute these commands in the PyMol's internal command line:

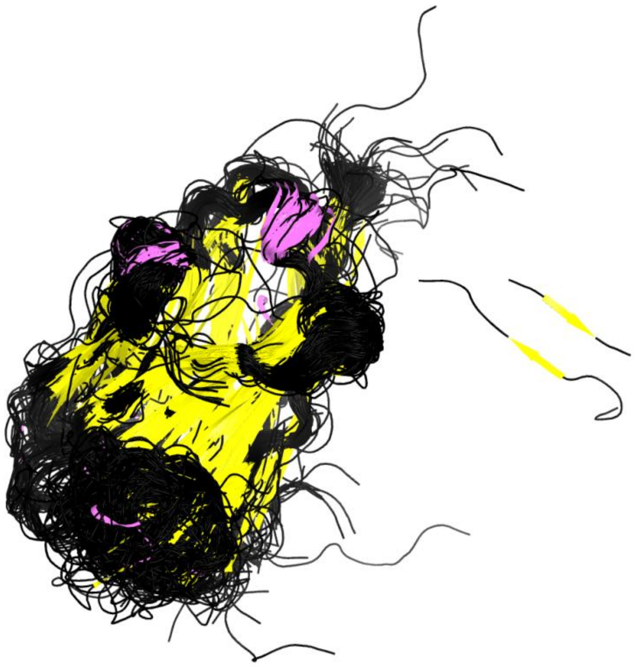
```
bg_color white
hide everything
bg_color white
set cartoon_discrete_colors, 1
set cartoon_oval_length, 0.3
set cartoon_oval_width, 0.035
set cartoon_loop_radius, 0.1
set cartoon_rect_width, 0.035
set cartoon_rect_length, 0.5

show cartoon
color black
color violet, ss h
color yellow, ss s
set ray_shadows, 0
```

These settings will provide the following view:



You can further use the ray command in PyMol to create high-quality illustrations of your structural superimposition (can take some time for a large number of structures):

```
set ambient, 0.5
ray 1280, 1024
```



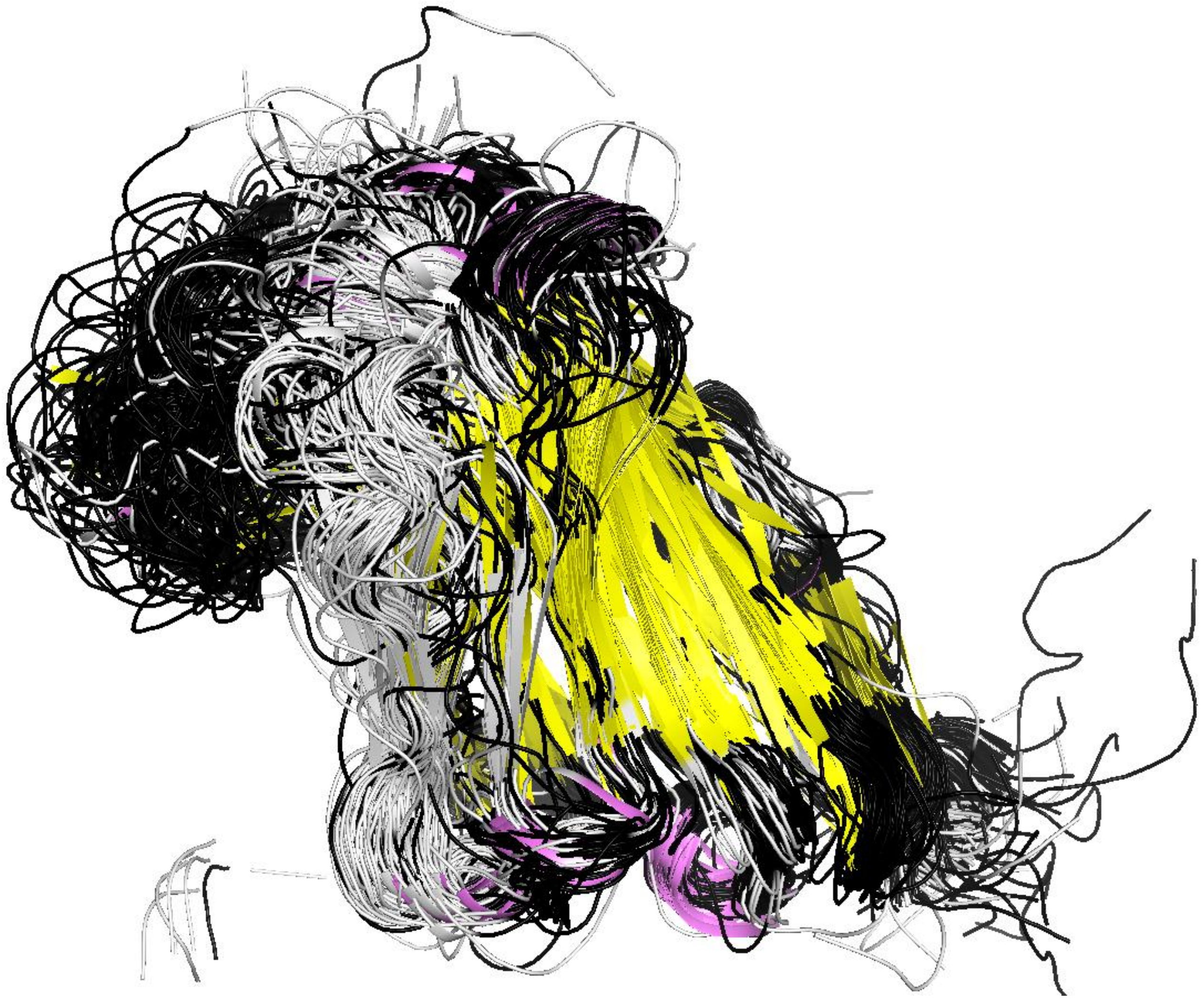
Analysis of the common structural core

On successful completion the parMATT/MATT produces, among other output, an instruction file for JMol (the “.spt” file) to highlight the common structural core of the aligned protein structures – i.e., structural equivalences which are shared by all proteins from the input set. This feature can help to compare evolutionarily related proteins and study the structure-function relationship. The JMol is a useful program, but it was not designed to handle large datasets of hundreds to thousands PDB structures. Therefore, it seems more appropriate to study the common structural core of the parMATT alignment by a much more versatile PyMol engine. To convert the “.spt” file with instructions for JMol into the PyMol format you can use the `jmol2pymol.pl` script. Below is a complete guide to using this script, explained on a particular example. Please note, that in this example 643 PDB files have to be loaded at once into PyMol. This operation should not be a problem for a modern Desktop computer, but may take a few minutes on a slower PC or a Notebook. Please be patient. As an alternative, delete as many PDB structures from the set and then run the PyMol to accelerate this demonstration.

Please note that THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND. PLEASE CHECK MANUALLY THE CONSISTENCY OF THE GENERATED OUTPUT.

```
#Download the example data
wget https://biokinet.belozersky.msu.ru/sites/default/data/Example_split.tar.gz
tar xzf Example_split.tar.gz

#Download the splitMATT2chains.sh script to parse the PDB output file
wget https://biokinet.belozersky.msu.ru/sites/default/data/splitMATT2chains
mv splitMATT2chains splitMATT2chains.sh
chmod 755 splitMATT2chains.sh

#Download the jmol2pymol.pl script to prepare PyMol instruction file
wget https://biokinet.belozersky.msu.ru/sites/default/data/jmol2pymol.pl
chmod 755 jmol2pymol.pl

#Enter the data folder
cd Example_split/

#Run the splitMATT2chains.sh to split the PDB output file into separate files
../splitMATT2chains.sh 2.60.40.10.pdb 2.60.40.10.txt split_pdbs

#Run the jmol2pymol.pl to convert the JMOL annotation file to the PyMol format
../jmol2pymol.pl 2.60.40.10.spt 2.60.40.10.txt 2.60.40.10.py

#Enter the folder with aligned PDB files
cd split_pdbs/
```

```
#Open all files in PyMol
pymol *pdb

#Now, in the PyMol’s command line run the PyMol annotation file
>run ../2.60.40.10.py
```



The listed commands will eventually produce the graphical output shown above. The common structural core is colored in pink (α-helixes), yellow (β-sheets), and black (loops), and the regions which are not part of the common structural core (are not equivalent in all input structures) are colored in white.

You can further use the ray command in PyMol to create high-quality illustrations of your structural superimposition (can take some time for a large number of structures):

```
set ambient, 0.5
ray 1280, 1024
```

The parMATT’s examples

The input and pre-calculated output files for these examples can be downloaded from the parMATT’s page at https://biokinet.belozersky.msu.ru/parmatt

Example 1 (small dataset)

This small dataset contains only 5 protein structures of highly structurally similar (>90% of matching secondary structure elements) MAP kinases. The multiple structural alignment of this set by parMATT should take just a few seconds on any modern hardware. Thus, the purpose of this data set is to quickly verify whether your build of parMATT’s binary from the source code was successful.

Download data: [download] (*Ctrl-click the link to initiate download in your browser*)

Execution. Extract the data from the downloaded archive, create the output folder, enter the input folder, and finally run the parMATT:

```
tar xzf Example_MAPK.tar.gz
cd Example_MAPK/
mkdir output_new
cd input
/path/to/parMatt -t 4 -L *pdb -o ../output_new/test
```

Your output files will be written to the `output_new` folder. You can check the pre-calculated output files for this example in the `output` folder of the extracted archive.

If the program fails to produce the output or takes a significant amount of time on this example you should revise the compilation procedure.

Example 2 (large dataset)

The purpose of the large dataset is to test scalability of parMATT on your multiprocessor system. The dataset should take ~2-5 hours to align on one CPU (depending on the CPU), and running parMATT on multiple nodes (many CPUs) should provide a significant speed-up. Before using the parMATT to do actual work you are advised to run the program several times with different resources, i.e., on 1 node/CPU, 2 nodes/CPUs, 4 nodes/CPUs, etc, and evaluate the scalability of your parMATT build on your particular hardware. The exact acceleration will depend on the particular configuration of your multi-processor system. However, if you do not experience any significant speedup at all, that would indicate a problem.

Download data: [download] (*Ctrl-click the link to initiate download in your browser*)

The input. The input set for this example is based on the **3.30.390.10** superfamily (Enolase-like, N-terminal domain) according to the CATH classification. The non-redundant set contains 111 protein structures in the PDB format with an average length of 130 amino acids.

Execution. Extract the data from the downloaded archive, create the output folder, enter the input folder, create the input list file, and finally run the parMATT:

```
tar xzf Example_3.30.390.10.tar.gz
cd Example_3.30.390.10/
mkdir output_new
cd input
find ./ -name "*pdb" > list.file
/path/to/parMatt -t 4 -L list.file -o ../output_new/3.30.390.10
```

Note, that the last command (i.e., the execution of the parMATT) will start parMATT on 4 physical cores of your local CPU and thus should be used only on your local desktop computer. To run on computing cluster or supercomputer please use a different command depending on the particular hardware and task manager (see the "Running parMATT" section above).

Running times. The amount of time required to calculate the alignment of the provided set will, of course, depend on the computing power implemented for the task, however will take hours by the order of magnitude:

- **1** x Intel Core i7-4790 3.60GHz (4 physical cores) ~ 4.6 hours;
- **1** x Intel Xeon CPU E5-2697 v3 2.60GHz (14 physical cores) ~ 2.2 hours;
- **4** x Intel Xeon CPU E5-2697 v3 2.60GHz (14 physical cores) ~ 0.7 hours;

In other words, running the parMATT on 4 nodes (4 CPUs) can provide x3.5 speed-up compared to the computational performance on one CPU. The exact acceleration will depend on the particular configuration of your multi-processor system. However, if you do not experience any significant speedup at all, that could mean one of the following:

- You are using the wrong command to launch the program on multiple nodes. Check the syntax of your MPI utility/queue manager;
- You did not use MPI compiler or it did not work. Revise your compilation procedure or contact your administrator for advise;

- Your hardware is not working properly. This is the most unlikely scenario. Try to troubleshoot the software part first (see above) and then contact your administrator as a last resort.

Output. Four files will be created on successful completion in the `output_new` folder:

- The `3.30.390.10.pdb` file with the coordinate representation of a multiple structural alignment, i.e., a PDB file with aligned coordinates of all 3D-models from the input;
- The `3.30.390.10.fasta` file with the sequence representation of a multiple alignment of the common structural core, i.e., a sequence alignment file in FASTA format;
- The `3.30.390.10.txt` file with a summary of the input PDBs (the pairwise comparison tree) and the output superimposition (number of residues in the core alignment, RMSD of the core alignment, the MATT's alignment quality score and the sequence representation of the common core alignment in the PHYLIP format);;
- The `3.30.390.10.spt` file with a Rasmol script to highlight aligned residues.

You can check the pre-calculated output files for this example in the `output` folder of the extracted archive.

Collecting a set of 3D-models of homologous proteins

The increasing number of protein structures in public databases provides new opportunities for comparative bioinformatic analysis of remote evolutionary relatives which have acquired new functions during natural selection and specialization from a common ancestor but preserved a common structural core. The parMATT can significantly accelerate the time-consuming process of building a structural alignment from a large collection of 3D-models of homologous proteins.

In general, there are three ways of collecting a set of functionally diverse homologous with a common structural organization.

- **Analysis of the literature.** Some protein superfamilies are well studied (e.g., the α/βhydrolase superfamily) and numerous scientific publications are available describing particular families and corresponding members with different functions. The respective three-dimensional models of these proteins can be manually collected from the PDB database. The advantage of collecting your PDB set by hand is that it will be based on experimental research and the corresponding functional annotation will be available for each included protein. The downside is that you can miss rare proteins whose properties have not been studied yet or were only poorly studied, as well as miss out protein structures recently added to the PDB but not yet described in the literature.
- **Structure similarity search.** Bioinformatic approach to collecting a set of remote homologs is based on detecting a significant structural similarity between the user-defined query protein and every protein structure available in the PDB. Selection of a query protein depends on the particular task and research objective. It can be the target protein selected for the further experimental design, or the most studied member of the superfamily. The **PDBeFold server** (http://www.ebi.ac.uk/msd-srv/ssm/) is available on-line and provides easy and intuitive interface to search for structure similarities in the PDB database. The advantage of collecting your PDB set by bioinformatic analysis is that all information available up to the date in public databases will be taken into account, including poorly studied and recently added proteins. The downside is that this process usually requires some knowledge of bioinformatics to collect and postprocess the set (e.g., eliminate redundant entries).
- **Mustguseal server**. **Mustguseal** is a bioinformatic protocol designed to build alignments of protein families and superfamilies, and a platform (a web-server) to provide a user-friendly web-based interface to the Mustguseal protocol through the World Wide Web (https://biokinet.belozersky.msu.ru/mustguseal).

Automatic collection and filtering of a non-redundant set of remote homologs by structure similarity search in the PDB database is part of the Protocol. Mustguseal can be used to collect a large set of proteins with diverse functions within a common structural core. This set can be downloaded by the user and aligned using the parMATT. The Mustguseal protocol and web-server is described in (Suplatov et al., *Bioinformatics* 2018).

Implementation of parMATT in the laboratory practice

Comparative bioinformatics is the cornerstone of computational approaches to understanding the sequence-structure-function relationship in proteins. Accurate alignment of protein families/superfamilies is crucial at studying structure-function relationship, but presents a methodological challenge due to low sequence similarity of evolutionarily distantly related homologues. Protein structure is more conserved throughout the evolution compared to sequence. It is therefore expected that three-dimensional alignment will provide more significant clues to protein function, properties and evolution than sequence alignment alone. Bioinformatic analysis of conserved and variable positions in large alignments of protein families/superfamilies can help with understanding of protein mechanisms, engineering enzymes with improved properties for practical application, and designing novel modulators of the activity of wild type proteins. The following three publications discuss these issues in more detail:

Suplatov, D., Kirilin, E., & Švedas, V. (2016). Bioinformatic Analysis of Protein Families to Select Function-Related Variable Positions. In Understanding Enzymes: Function, Design, Engineering, and Analysis (pp. 351-385) *Ed. Allan Svendsen.* Pan Stanford. [link]

Suplatov, D., Voevodin, V., & Švedas, V. (2015). Robust enzyme design: Bioinformatic tools for improved protein stability. *Biotechnology journal*, 10(3), 344-355. [link]

Suplatov, D., & Švedas, V. (2015). Study of functional and allosteric sites in protein superfamilies. *Acta Naturae*, 7(4), 27, 34-45. [link]

The output alignment of the parMATT is compatible with Modes 2 and 3 of the **Mustguseal server**. This user-built core structural alignment can be submitted to the Mustguseal server to build a larger structure-guided sequence alignment of the corresponding superfamily by incorporating all available information about sequences of homologous proteins from public databases.

The parMATT's license

The parMATT software is licensed under the GNU public license version 2.0.

Citing parMATT

If you find parMATT or its results useful please cite our work:

Shegay M., Suplatov D., Popova N., Švedas V., Voevodin Vl. (2019) parMATT: Parallel multiple alignment of protein 3D-structures with translations and twists for distributed-memory systems, Bioinformatics DOI:10.1093/bioinformatics/btz224

The parMATT is based on the MATT algorithm and code:

Menke, M., Berger, B., & Cowen, L. (2008). Matt: local flexibility aids protein multiple structure alignment. *PLoS Comput Biol.*, 4(1), e10.