CoMIC (Coevolution via MI on CUDA)
Short Documentation

Michael Waechter, Kathrin Jaeger, Daniel Thuerck, Stephanie Weissgraeber,
Sven Widmer, Michael Goesele, and Kay Hamacher

November 16, 2012

The CoMIC source code as well as this documentation is available on the project's web page:

`http://www.gris.tu-darmstadt.de/projects/comic/`.

Contents

1 Version History

Version	Date	Description
1.2	Nov. 16, 2012	Code compiles on CUDA 5.0 and 4.x
1.1	Aug. 23, 2012	Reduced CPU RAM and GPU global memory usage to approximately one half for 2-point and one sixth for 3-point MI
1.0	Apr. 17, 2012	Initial Release. Computation of 2-point and 3-point MI, $E(\mathrm{MI})$, $E(\mathrm{MI}^2)$ and percentiles

2 Compilation

2.1 Prerequisites

Please note that we only support compilation on Linux-based systems.

CoMIC requires an NVIDIA GPU of at least CUDA compute capability 2.0. You can check your GPU's compute capability at this web page: `http://developer.nvidia.com/cuda-gpus`. We successfully ran the project on systems with a Tesla C2070, GeForce GTX 460, 480, 560 and 590.

We recommend that the GPU with the most compute power is installed as secondary GPU with a slower one installed as primary GPU. The software automatically chooses the faster GPU as compute device. In case you are trying to compute three-point MI with the software, an individual GPU function call might take longer than 8 seconds, which automatically leads to the launch being terminated by the X server, if the GPU you are computing on is the primary device.

You need to have

- `cmake`, `make` and `gcc`

- the newest NVIDIA drivers for your GPU
- the CUDA toolkit (version 4.0 or newer, latest tested version: 5.0)
- and the CUDA SDK (only for versions 4.x)

installed.

You can find the CUDA toolkit, the CUDA SDK and information on how to install both on `http://developer.nvidia.com/cuda-downloads`. Please follow the "Download CUDA Toolkit Production Release" link, then follow the "documentation" link and download the "CUDA Getting Started Guide (Linux)". This guide describes the toolkit's as well as the SDK's installation. Please follow it closely as it is easy to forget some of the steps, like setting the `LD_LIBRARY_PATH` or compiling the SDK. Please also make sure that you install the toolkit as well as the SDK into the same paths as described in the guide, as these paths have been verified to work during compilation.

2.2 Compilation

1. Download the whole source code from the project's homepage `http://www.gris.tu-darmstadt.de/projects/comic/` and unzip it:

   ```
   cd path/to/where/you/downloaded/the/code
   tar -xvzf CoMIC.tar.gz
   ```

2. Change into the unzipped source folder, create a `build` subfolder and change into it:

   ```
   cd src
   mkdir build
   cd build
   ```

3. Run "`cmake ..`". This requires the `CMakeLists.txt` to be in the folder above the `build` folder. If you created the `build` folder elsewhere, the path handed over to `cmake` needs to be changed accordingly.

4. Run "`make`". If this should not work, run "`ccmake .`", go to the "`CUDA_SDK_ROOT_DIR`" line (if you cannot find this line, you may have to hit "`t`" first to toggle to advanced mode), hit return, enter the path to where you installed the CUDA SDK (`/usr/local/cuda_sdk`, `~/cuda_sdk` or the like), hit return again, then "`c`" and then "`g`"

3 Execution

3.1 Input file format

The `CoMIC` program takes a multiple sequence alignment in the FASTA format as input.

In the FASTA format each sequence starts with a greater-than (">") symbol followed by the name or description of the sequence. This is followed by a line break, followed by the sequence data itself. The sequence data itself consists of characters for the different amino / nucleic acids and can be interrupted by line breaks for aesthetic reasons. After the sequence data there is another line break followed by another ">", the description and the data for the next sequence.

An example file (from `http://en.wikipedia.org/w/index.php?title=FASTA_format&oldid=467031641`) looks like this:

```
>SEQUENCE_1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDFDKAVQLLREKGLGKAAKKADRLAAEG
LVSVKVSDDFTIAAMRPSYLSYEDLDMTFVENEYKALVAELEKENEERRRLKDPNKPEHK
IPQFASRKQLSDAILKEAEEKIKEELKAQGKPEKIWDNIIPGKMNSFIADNSQLDSKLTL
MGQFYVMDDKKTVEQVIAEKEKEFGGKIKIVEFICFEVGEGLEKKTEDFAAEVAAQL
>SEQUENCE_2
SATVSEINSETDFVAKNDQFIALTKDTTAHIQSNSLQSVEELHSSTINGVKFEEYLKSQI
ATIGENLVVRRFATLKAGANGVVNGYIHTNGRVGVVIAAACDSAEVASKSRDLLRQICMH
....
```

A longer description of the FASTA format can be found in `http://en.wikipedia.org/w/index.php?title=FASTA_format&oldid=467031641`.

Note, that CoMIC only supports "X" as "any character", "-" as gap character and the standard 22 amino acid characters A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W and Y. All other characters will automatically be set to "X". In this case the software gives a warning and continues the computation.

3.2 Command-line parameters

The compiled program has the following command-line options:

- `-i=<infile>` is the only obligatory parameter and specifies the FASTA input file on which computations shall be performed.
- `-it=<number of iterations>` defines the number of shuffle null model iterations the program shall perform. The default is 10,000.
- `-dna` should be used, if the input file represents DNA/RNA data instead of proteins. The default is proteins. This option speeds up computation considerably.
- `-cpu` makes the program do its computations on the CPU instead of the GPU.
- `-threepoint` computes 3-point instead of 2-point MI values.
- `-uppertriangle` lets the program output the whole MI matrix. By default it only ouputs the lower triangle (or the lower sixth in case of 3-point MI) because the upper triangle's content is equivalent.
- `-percentiles` computes percentiles for all MI values instead of $\langle \widetilde{\mathrm{MI}}_{ij} \rangle$ and $\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle$ values.

3.3 Sample execution

As the program takes a FASTA file as input, you need to obtain one for testing first.

If the code is already compiled (see Sec. 2.2), you have some sequence alignment file available and your current working directory is the `build` folder, a program call could be:

```
./comic -i=path/to/some_fasta_file.fasta -it=2
```

It computes 2-point MI on the GPU with 2 Shuffle Null-Model iterations[1] and outputs the $\langle \widetilde{\mathrm{MI}}_{ij} \rangle$ and $\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle$ matrices' lower triangles.

Please note, that 2 iterations are only performed here for testing purposes, as for large sequence alignments computation may take extremely long[2]. For stastically meaningful results, it is necessary to compute in the order of 10,000 iterations. Computing 2 (or more) iterations first can give you an estimate of how long a full 10,000 iteration computation will take.

Other example calls could be

```
./comic -i=path/to/some_fasta_file.fasta -cpu -it=1000
```

for computing 2-point MI with 1,000 iterations on the CPU instead of the GPU, or

```
./comic -i=path/to/some_fasta_file.fasta -threepoint -it=1000
```

for computing 3-point MI with 1,000 iterations on the GPU.

3.4 Output file format

The program outputs three different matrices to the path, where the input file is located:

- `<infile>.MI.out` contains the unmodified MI values.
- `<infile>.E(MI).out` contains the $\langle \widetilde{\mathrm{MI}}_{ij} \rangle$ values (the MI averaged over all shuffle null model iterations).
- `<infile>.E(MI^2).out` contains the $\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle$ values (the squared MI averaged over all shuffle null model iterations).

[1]2 iterations is far from being statistically sufficient, but it should be enough for a simple test run.

[2]in the order of days for 10,000 iterations even for normal two-point MI

If you compute 3-point MI (by using the `-threepoint` option), the output files' names will be changed to `<infile>.3point.MI.out`, `<infile>.3point.E(MI).out` and `<infile>.3point.E(MI^2).out` respectively.

If you compute percentiles (by using the `-percentiles` option), the output files' names will be changed to `<infile>.percentiles.out` or `<infile>.3point.percentiles.out` respectively.

The output files' format will look like this:

```
i=0 j=0 k=0:  2.07879
i=0 j=0 k=1:  1.17107
i=0 j=1 k=1:  0.847907
i=1 j=1 k=1:  1.43245
i=0 j=0 k=2:  1.52182
i=0 j=1 k=2:  0.83057
i=1 j=1 k=2:  0.925954
i=0 j=2 k=2:  2.42419
i=1 j=2 k=2:  2.15149
i=2 j=2 k=2:  3.88352
...
```

Note, that the order of the coordinates does not matter as for example $\mathrm{MI}_{1,2,3} = \mathrm{MI}_{2,1,3} = \mathrm{MI}_{1,3,2} = \mathrm{MI}_{3,1,2} = \mathrm{MI}_{2,3,1} = \mathrm{MI}_{3,2,1}$ holds. This is the reason why by default the program outputs only the matrices' lower triangle for 2-point MI or lower sixth for 3-point MI.

4 Limitations

Concerning the sequences' lengths or the number of sequences in the multiple sequence alignment we are not aware of limitations other than CPU and GPU memory space: During program execution three matrices of size $\sim \frac{1}{2} \cdot L^2 \cdot \mathrm{sizeof(float)}$ for 2-point and $\sim \frac{1}{6} \cdot L^3 \cdot \mathrm{sizeof(float)}$ for 3-point MI computation (with L being the sequence length) need to fit into CPU and GPU memory.
For 2-point MI this limitation is probably not an issue, but for 3-point MI it might be one.

5 Known Issues

"**cudaSafeCall() Runtime API error 6: the launch timed out and was terminated.**" occurs, when the computation of one single Shuffle Null Model iteration takes longer than eight seconds. This is especially likely if you try to compute 3-point MI or if you are working on extremely large FASTA files. In this case you should consider installing your GPU with the most compute power (which will automatically be chosen as compute device) as your computer's secondary GPU and installing a slower one as primary GPU, because the X server automatically terminates jobs on the primary GPU, that take longer than 8 seconds.

"**terminate called after throwing an instance of 'std::bad_alloc'**" may occur, when computing 3-point MI on files with rather long sequence lengths. In this case it is possible that the three-dimensional MI matrices do not fit into CPU or GPU memory.