FM3VCF Instruction

Zhen Zuo

2023-6-7

Index

1. FM3vcf

1.1 Compiling

Required library: gcc; zlib; pthread; openmp

For Linux
Go to the directory of Linux version
cd zM3vcf

Compile vcf reading lib to the specified directory
make vcflib

To compile and generate the zM3vcf tool and the m3vcf interface testing program (m3vcfTest):
make

For Windows
To begin, you need to install the tdm64-gcc compiler. You can download it from the website: https://jmeubank.github.io/tdm-gcc/. Alternatively, you can find the pre-downloaded installer package named tdm64-gcc-10.3.0-2.exe in the "win_supportPackage" folder. During the installation, make sure to select the option to install openMP. This is necessary for compiling the project correctly. For detailed installation instructions, please refer to Appendix 1.
Additionally, you need to download the zlib library and compile it using tdm64-gcc. You can download the zlib source code from https://www.zlib.net/. Alternatively, you can find the pre-downloaded source code package named zlib-1.2.13.tar.gz in the "win_supportPackage" folder. After successful compilation, copy the compiled zconf.h and zlib.h files to the include folder in the tdm64-gcc installation directory. Similarly, copy the compiled libz.a file to the lib folder in the tdm64-gcc installation directory. Detailed installation instructions can be found in Appendix 1.
Next, open the command prompt and navigate to the project folder. Access the Windows version directory:
cd zM3vcf_win

To compile the vcflib library and generate the vclib library files to the specified directory:
mingw32-make vcflib

To compile and generate the zM3vcf.exe command line tool and the m3vcf interface testing program (m3vcfTest.exe):

```
mingw32-make
```

1.2 Command line

For Linux

Compress vcf to m3vcf

The parameter -b specifies the number of records in each compression block. The default value is 1000.

The parameter -o specifies the output file name.

The parameter -O specifies the output file type. Use "M" for uncompressed and "m" for compressed.

The parameter -t allows you to specify the maximum number of pthreads (read, write, compress). The value must be at least 3. The default is 8, including 1 read thread, 1 write thread, and 6 compress threads. Note that this thread count does not include the openMP threads used by vcflib.

The parameter -m allows you to specify the maximum amount of memory in GB. It is recommended not to specify this parameter.

Convert m3vcf to vcf

Use the -o parameter to specify the output file name.

Use the -O parameter to specify the output file type. Use "M" for uncompressed and "m" for compressed.

For Windows

The functionality and parameters remain the same as in Linux on the Windows platform.

1.3 The example of command line

To use the zM3vcf command-line tool in the project folder to compress the VCF file located in the "testFile" folder, which is derived from the first 10 samples of the first 20 markers from the file "ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20110521/ALL.chr22.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz", and save it as "ALL.chr22.20Markers.10Samples.m3vcf.gz" in the "testFile" directory, you can execute the following commands:

For Linux:

```
./zM3vcf compress testFile/ALL.chr22.20Markers.10Samples.vcf.gz -O m -o testFile/ALL.chr22.20Markers.10Samples.m3vcf.gz
```

For Windows:

```
zM3vcf.exe compress testFile\ALL.chr22.20Markers.10Samples.vcf.gz -O m -o testFile\ALL.chr22.20Markers.10Samples.m3vcf.gz
```

To use the zM3vcf command-line tool in the project folder to convert the previously generated m3vcf file, "ALL.chr22.20Markers.10Samples.m3vcf.gz," located in the "testFile" folder, back to a VCF file named "ALL.chr22.20Markers.10Samples.vcf.gz" in the current working directory, you can execute the following commands:
For Linux:

```
./zM3vcf convert testFile/ALL.chr22.20Markers.10Samples.m3vcf.gz -O m -o ALL.chr22.20Markers.10Samples.vcf.gz
```

For Windows:

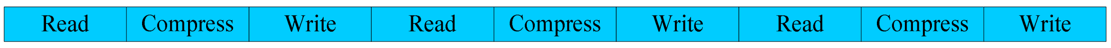
```
zM3vcf.exe convert testFile\ALL.chr22.20Markers.10Samples.m3vcf.gz -O m -o ALL.chr22.20Markers.10Samples.vcf.gz
```

1.4 Test compiled file of interface

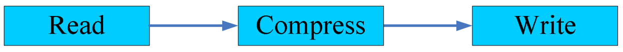
The m3vcfTest program (m3vcfTest.exe on the Windows platform) in the project folder demonstrates the entire process of compressing the VCF file named "ALL.chr22.20Markers.10Samples.vcf.gz" located in the "testFile" folder to the m3vcf file named "ALL.chr22.20Markers.10Samples.m3vcf.gz" in the same folder using the interface functions from the m3vcf library. It also showcases the process of decompressing and restoring the generated m3vcf file, "ALL.chr22.20Markers.10Samples.m3vcf.gz," back to the VCF file named "ALL.chr22.20Markers.10Samples.IF.vcf.gz" in the "testFile" folder.
By using this test program, users can learn how to utilize the interface functions to efficiently and conveniently perform high-speed conversions between VCF and m3vcf files in their own code.

1.5 The mechanism of software acceleration

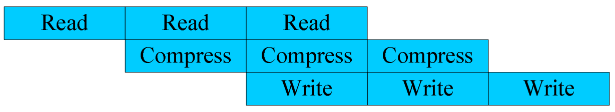
For M3vcftools, the main steps involved in completing a compression task are as follows:
Read: Read data from the VCF file and parse it into the corresponding data structures in memory. Compress: Convert VCF file records into compressed m3vcf file records.
Write: Write the generated data into the m3vcf file.
Since the original program is single-threaded, the process of performing three compression tasks within a single core of the CPU is as follows:



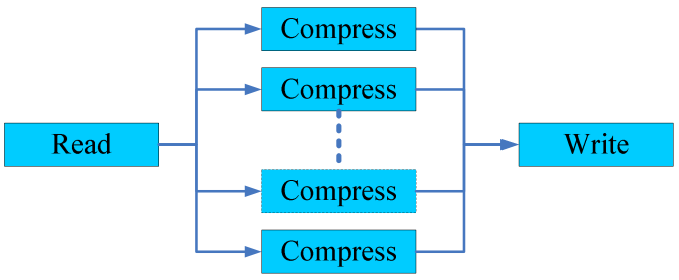
For FM3vcf, the first step is to separate the Read, Compress, and Write steps and assign them to different threads to complete the process independently.



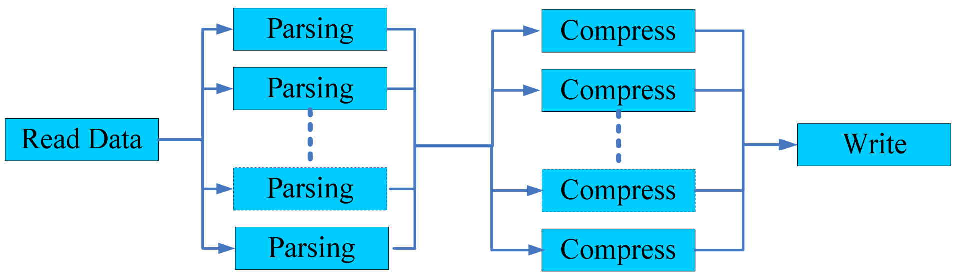
Therefore, to complete the same three compression tasks in parallel within multiple cores of the CPU, the process is as follows:



By comparing the vertical comparison with the previous diagram, it can be observed that for the same three compression tasks, the original program requires 9 steps, while the current program only requires 5 steps to complete. This results in a significant improvement in speed, especially as the number of compressions increases. This method allows for a partial speedup.
Through experimentation, it has been determined that among the Read, Compress, and Write steps, the Compress step takes the longest time. Therefore, a multi-threading approach using pthread is employed to run the Compress step in parallel across multiple cores of the CPU. This approach aims to ensure that the time taken for the Read, Compress, and Write steps is approximately equal and well-matched.



This method can further enhance the speed improvement.
Within the Read step, it can be further divided into two parts:
Read Data: Reading the data from the disk into memory.
Parsing: Parsing the raw data from memory into corresponding data structures.
The Parsing part consumes a significant amount of time within the Read step. To accelerate this process, an OpenMP multi-threading approach is employed specifically for the Parsing part.



The software utilizes our vcf reading library to handle this functionality. By implementing this approach, it is possible to achieve further acceleration in the Read step.

2. VCF reading

The main functionality of the vcflib library is to efficiently read VCF files into memory. It supports both uncompressed VCF files and gzipped files compressed with zlib.

2.1 Compiling

For Linux
Go to the directory of Linux version

```
cd zM3vcf/vcflib
```

Compile vcf reading lib to the specified directory

```
make
```

For Windows
The command lines remain the same as in Linux on the Windows platform

2.2 Files introduction

Makefile: The compilation rules file for the library, which is used to compile the library using the "make" command. bin: This directory stores the relevant files generated after successful compilation of the library. (Empty before compilation) - vcflib.h: The header file of the library, which includes declarations of all the function interfaces provided by the library, as well as related user-adjustable macro parameters. - libvcf.so: The dynamic library file of the library. - libvcf.a: The static library file of the library. inc: This directory stores the header files of the library. - baseTool.h: Declarations of internal basic utility functions used in the library. -

vcflib.h: The header file of the library, which includes declarations of all the function interfaces provided by the library, as well as related user-adjustable macro parameters. obj: This directory stores the intermediate files generated during the compilation process of the library. (Empty before compilation) src: This directory stores the source code files of the library. - baseTools.c: Internal basic utility functions used in the library. - vcfFile.c: The main functional interface functions of the library. tst: This directory stores example code for the library, including a program called "simpleRead" that demonstrates the simplest read operation.

2.3 Interface

(1) File Opening and Closing Interfaces

VCF_STATUS vcfFileOpen(VCF_FILE *fp, const char *fileName, FILE_MODE fileMode, unsigned int parseItem);

Function:

Opens an existing VCF file in read mode. It supports uncompressed text file format and gzipped format compressed with zlib. It also enables logging for the vcflib library.

Parameters:

fp: A pointer to the VCF_FILE structure. It is the file handle in the vcflib library. The structure is defined as follows:

```
typedef struct
{
    union
    {
        FILE * fp;
        gzFile gfp;
    }fp;
    FILE_MODE mode;
    FILE_HEAD head;
    int numSamples;
    unsigned int parsingItems;
}VCF_FILE;
```

fileName: The path and name of the VCF file to be opened.

fileMode: Specifies the compression format of the file to be opened. FILE_MODE_NORMAL indicates uncompressed format, while FILE_MODE_GZ indicates zlib-compressed gz format.

parseItem: Specifies which fields to parse when reading the file. P_DS|P_GT indicates that both DS and GT fields will be parsed, while P_GT indicates that only the GT field will be parsed.
Return Values:
VCF_ERROR: Failed to open the file.
VCF_OK: File opened successfully.

VCF_STATUS vcfFileCreate(VCF_FILE *fp,const char *fileName,FILE_MODE fileMode);
Function:
Creates a VCF file in write mode. If the file already exists, its contents will be cleared. Supports uncompressed text file format and zlib-compressed gz format. Enables vcflib logging.
Parameters:
fp: Pointer to the VCF_FILE structure; see details above (vcfFileOpen).
fileName: The path and name of the VCF file to be created/opened. fileMode: Specifies the compression format of the file to be created/opened. FILE_MODE_NORMAL indicates uncompressed format, while FILE_MODE_GZ indicates zlib-compressed gz format.
Return Values:
VCF_ERROR: Failed to create/open the file.
VCF_OK: File created/opened successfully.

VCF_STATUS vcfFileAppend(VCF_FILE *fp,const char *fileName,FILE_MODE fileMode);
Function:
Creates a VCF file in write mode. If the file already exists, its contents will be preserved, and the file pointer will be positioned at the end of the file. Supports uncompressed text file format and zlib-compressed gz format. Enables vcflib logging.
Parameters:
fp: Pointer to the VCF_FILE structure; see details above (vcfFileOpen).
fileName: The path and name of the VCF file to be created/opened. fileMode: Specifies the compression format of the file to be created/opened. FILE_MODE_NORMAL indicates uncompressed format, while FILE_MODE_GZ indicates zlib-compressed gz format.
Return Values:
VCF_ERROR: Failed to append/open the file.
VCF_OK: File appended/opened successfully.

VCF_STATUS vcfFileClose(VCF_FILE *fp);
Function:

Closes the opened file, clears and releases the memory occupied by the file header, and closes the vcflib logging.
Parameters:
fp: Pointer to the VCF_FILE structure; see details above (vcfFileOpen).
Return Values:
VCF_ERROR: Failed to close the file.
VCF_OK: File closed successfully.

(2) File Reading Interfaces

VCF_STATUS vcfFileReadLine(VCF_FILE *fp,char *lineStr,int lineSize);
Function:
Reads the current line from the opened VCF file pointed to by fp and stores the content in lineStr.
Parameters:
fp: Pointer to the VCF_FILE structure.
lineStr: Memory pointer to store the read file content.
lineSize: Length of the memory pointed to by lineStr.
Return values:
VCF_ERROR: Failed to read.
VCF_OK: Read successful.

VCF_STATUS vcfFileReadHead(VCF_FILE *fp);
Function:
Reads the VCF file header section from the opened VCF file pointed to by fp and stores the content in the FILE_HEAD structure pointed to by fp->head.
Parameters:
fp: Pointer to the VCF_FILE structure. The member variable head of type FILE_HEAD is used to store the file header of the VCF file. The structure is defined as follows:

```
typedef struct
{
        char **metaInfoLines;
        int numMetaInfoLines;
        char *headerLine;
}FILE_HEAD;
```

Return values:
VCF_ERROR: Failed to read.
VCF_OK: Read successful.

VCF_STATUS getNumMetaInfoLines(FILE_HEAD *fhp,int *NumMetaInfoLines);

Function:

Get the number of meta information lines from the vcf file header pointed to by fhp.

Parameters:

fhp: Pointer to the FILE_HEAD structure. See above for details (vcfFileReadHead).

NumMetaInfoLines: The number of meta information lines obtained from the vcf file header pointed to by fhp.

Return values:

VCF_ERROR: Failed to read.

VCF_OK: Read successful.

VCF_STATUS getNumSamples(VCF_FILE *fp,int *NumSamples);

Function:

Get the number of samples in the vcf file pointed to by fp.

Parameters:

fp: Pointer to the VCF_FILE structure.

NumSamples: The number of samples obtained from the vcf file.

Return values:

VCF_ERROR: Failed to read.

VCF_OK: Read successful.

char* vcfFileParseDataLineInfo(char *lineStr,DATA_INFO *dataInfo);

Function:

Parse the first nine items of a line of vcf file data in lineStr and store them in the DATA_INFO structure pointed to by dataInfo. After the operation, the first nine items are removed from lineStr.

Parameters:

lineStr: A line of data from a vcf file.

dataInfo: Pointer to the DATA_INFO structure. The DATA_INFO structure in the vcflib library is used to store the first nine items of each line of data in a vcf file. The structure is defined as follows:

```
typedef struct
{
        char *chrom;
         char *pos;
         char *ID;
         char *ref;
        char *alt;
        char *qual;
        char *filter;
        char *info;
        char *format;
}DATA_INFO;
```

Return value:
The vcf file data line with the first nine columns removed.

VCF_STATUS vcfFileParseDataLine(VCF_FILE *fp,char *lineStr,DATA_LINE *dlp);
Function:
Parse a line of VCF file data from lineStr and store it in the dlp pointer to the DATA_LINE structure.
Parameters:
fp: Pointer to the VCF_FILE structure.
lineStr: Line of VCF file data.
dlp: Pointer to the DATA_LINE structure. It is used to store the data from the line of VCF file data.
The structure is defined as follows:

```
typedef struct
{
        char *rawDataLine;
        DATA_INFO dataInfo;
        char *samplesRawString;
        char *gtData;
        float *dsData;
        int numSamples;
}DATA_LINE;
```

Return Value:
VCF_ERROR: Parsing failed.
VCF_OK: Parsing succeeded.

VCF_STATUS vcfFileReadDataLine(VCF_FILE *fp,DATA_LINE *dlp);
Function:
Reads the current data line from the opened VCF file pointed to by fp and stores the content in the DATA_LINE structure pointed to by dlp.
Parameters:
fp: Pointer to the VCF_FILE structure.
dlp: Pointer to the DATA_LINE structure. Refer to the explanation above for details on the structure.
Return Value:
VCF_ERROR: Failed to read.

VCF_OK: Read successfully.

VCF_STATUS vcfFileReadDataBlock(VCF_FILE *fp,DATA_BLOCK *dbp,int numLines);

Function:

Reads numLines data lines from the opened VCF file pointed to by fp and stores the content in the DATA_BLOCK structure pointed to by dbp.

Parameters:

fp: Pointer to the VCF_FILE structure.

dbp: Pointer to the DATA_BLOCK structure. The structure is defined as follows:

```
typedef struct
{
        DATA_LINE *dataLines;
        int numDataLines;
}DATA_BLOCK;
```

numLines: Number of lines to read from the file at once.

Return Value:

VCF_ERROR: Failed to read.

VCF_OK: Read successfully.

VCF_STATUS vcfFileReadDataBlockOverlap1Line(VCF_FILE *fp,DATA_BLOCK *dbp,int numLines);

Function:

Reads numLines data lines from the opened VCF file (fp) and stores the content in the DATA_BLOCK structure pointed to by dbp. Except for the first time, each subsequent data block read will have an overlap of one line, where the first line of the new block is the last line of the previous block.

Parameters:

fp: Pointer to the VCF_FILE structure.

dbp: Pointer to the DATA_BLOCK structure. See above for its definition.

numLines: Number of lines to read from the file at once.

Return Value:

VCF_ERROR: Failed to read.

VCF_OK: Read successfully.

(3) File Writing Interfaces

VCF_STATUS vcfFileWriteLine(VCF_FILE *fp,char *lineStr);

Function:

Writes the content of lineStr as a line to the opened VCF file (fp).

Parameters:
fp: Pointer to the VCF_FILE structure.
lineStr: Pointer to the memory where the content to be written is stored.
Return Value:
VCF_ERROR: Failed to write.
VCF_OK: Write successful.

VCF_STATUS vcfFileWriteHead(VCF_FILE *fp,FILE_HEAD *fhp);
Function:
Writes the VCF file header information from the fhp structure to the opened VCF file fp.
Parameters:
fp: Pointer to the VCF_FILE structure.
fhp: Pointer to the FILE_HEAD structure containing the VCF file header information.
Return Value:
VCF_ERROR: Failed to write.
VCF_OK: Write successful.

VCF_STATUS vcfFileAddMetaInfoLine(FILE_HEAD *fhp,int posIndex,char *MetaInfoLine);
Function:
Adds the MetaInfoLine to the VCF file header structure pointed to by fhp, specifying the position of the added MetaInfoLine as the posIndex line within the meta information.
Parameters:
fhp: Pointer to the FILE_HEAD structure.
posIndex: Index specifying the position of the added meta information line within the meta information.
MetaInfoLine: Meta information line to be added.
Return Value:
VCF_ERROR: Failed to write.
VCF_OK: Write successful.

VCF_STATUS vcfFileRemoveMetaInfoLine(FILE_HEAD *fhp,int posIndex);
Function:
Removes the meta information line at position posIndex from the VCF file header structure pointed to by fhp.
Parameters:
fhp: Pointer to the FILE_HEAD structure.
posIndex: Index of the meta information line to be removed.
Return Value:
VCF_ERROR: Failed to delete.

VCF_OK: Deletion successful.

(4) Other Interfaces

void clearFileHead(FILE_HEAD *fhp);
Function:
Clears the data in the FILE_HEAD structure pointed to by fhp and frees its memory space.
Parameters:
fhp: Pointer to the FILE_HEAD structure.
Return Value:
None

void clearDataLine(DATA_LINE *dlp);
Function:
Clears the data in the DATA_LINE structure pointed to by dlp and frees its memory space. (Note: If vcfFileReadDataLine is called with the same dlp parameter multiple times, you need to call this function to clear and free the content of the previous DATA_LINE structure before calling vcfFileReadDataLine again, otherwise it may cause memory leaks.)
Parameters:
dlp: Pointer to the DATA_LINE structure.
Return Value:
None

void clearDataBlock(DATA_BLOCK *dbp);
Function:
Clears the data in the DATA_BLOCK structure pointed to by dbp and frees its memory space.
Parameters:
dbp: Pointer to the DATA_BLOCK structure.
Return Value:
None

VCF_STATUS vcfPopSubString(char **lineStr,char *subStr);
Function:
Extracts the first segment substring delimited by spaces, tabs ('\t'), or newline ('\n') from the string pointed to by lineStr and stores it in subStr. After the operation, the extracted substring is removed from lineStr.
Parameters:
lineStr: Pointer to the long string to be truncated.
subStr: Pointer to store the extracted substring.

Return Value:
VCF_ERROR: Failed to extract the substring; or the long string is empty (all segments have been extracted).
VCF_OK: Substring extraction successful.

void printDataLine(DATA_LINE *dlp);
Function:
Prints and displays the values of all member variables in the DATA_LINE structure pointed to by dlp. This is useful for visualizing the content stored in dlp.
Parameters:
dlp: Pointer to the DATA_LINE structure.
Return Value:
None.

2.4 Adjustable macro

The vcflib library provides some macro definitions in the inc/vcflib.h file that users can adjust according to their specific needs.

BT_MAX_LINE_SIZE: This macro defines the maximum number of characters allowed in a single line when reading a VCF file. The current default value is 6 million. If you encounter errors due to extremely long lines in the VCF file you're processing, you can increase this value and recompile the library.

Example usage: **#define BT_MAX_LINE_SIZE 8000000**

OPEN_MP_THREAD_NUM: This macro determines the number of OpenMP threads used for parsing VCF file data. The current default value is 10. By increasing this value, you can potentially improve the overall speed of reading VCF files using the library, subject to the limitations of disk I/O and the number of available CPU cores. You can increase this value and recompile the library.

Example usage: **#define OPEN_MP_THREAD_NUM 16**

Please note that modifying these macro values should be done with caution and based on your specific requirements and system resources. The new library must be re-compiled after modifying these macro values.

3. Tools

In the FM3vcf folder, there are several accompanying tools provided for testing the speed and efficiency of the project. These tools can assist you in evaluating and comparing performance under different configurations or optimization strategies. Here is a brief introduction to some of these tools:

3.1 Compiling

Here are the instructions for compiling the project on different platforms:
For Linux:
Navigate to the project folder and enter the Linux version directory:

```
cd zM3vcf
```

Compile the vcflib library and generate the library file in the specified directory:

```
make vcflib
```

Enter the tools folder:

```
cd tools
```

Compile and generate the small tools:

```
make
```

For Windows
Open the command prompt and navigate to the project folder, then enter the Windows version directory:

```
cd zM3vcf_win
```

Compile the vcflib library and generate the library file in the specified directory:

```
mingw32-make vcflib
```

Enter the tools folder:

```
cd tools
```

Compile and generate the small tools:

```
mingw32-make
```

3.2 Instruction

(1) pVcf

The source code for the tool is copyCreateVcf.c, located in the tools folder. Its functionality is to create a new VCF test file by populating it with the content of a real VCF file, multiplied by a factor of 'n'. Each line of the new VCF file is composed by randomly shuffling the content of 'n' lines from the original VCF file. To use this tool, you will need a real VCF file.
The usage is as follows:

```
pVcf sourceVCFFile sampleTimes targetVCFFile
```

Where:
sourceVCFFile is the path to the original VCF file.
sampleTimes is the multiplication factor for expanding the original VCF file.
targetVCFFile is the path to the generated new VCF file.
For example, to use the tool to double the number of samples in the "ALL.chr22.20Markers.10Samples.vcf.gz" file located in the "testFile" folder and generate a new file named "ALL.chr22.20Markers.20Samples.vcf.gz" in the same folder, you can use the following command (assuming the current working directory is the "tools" folder):

```
./pVcf ../testFile/ ALL.chr22.20Markers.10Samples.vcf.gz 2 ../testFile/
ALL.chr22.20Markers.20Samples.vcf.gz
```

（For windows is same as Linux）
(2) gVMH
The source code for this tool is getVcfMH.c, located in the tools folder. Its purpose is to print the number of markers and samples in a specified VCF file, and test if each marker in the VCF file complies with the expected number of samples.
The usage of this tool is as follows:

```
gVMH VCFFile
```

Where:
VCFFile is the path to the VCF file you want to view.
For example, to use this tool to print the number of samples and markers in the file ALL.chr22.20Markers.10Samples.vcf.gz located in the "testFile" folder, and check if each marker in every line has consistent sample values, the command would be as follows (assuming the current working directory is the tools folder):

```
./gVMH ../testFile/ ALL.chr22.20Markers.10Samples.vcf.gz
```

（For windows is same as Linux）
(3) vCM
The source code is located in the tools folder, and it is named vcfCutMarkers.c. It is used to extract the first n markers and the first m samples from a specified large VCF file and generate a new smaller VCF file.
Usage:

vCM sourceVCFFile targetVCFFile markersNumber samplesNumber

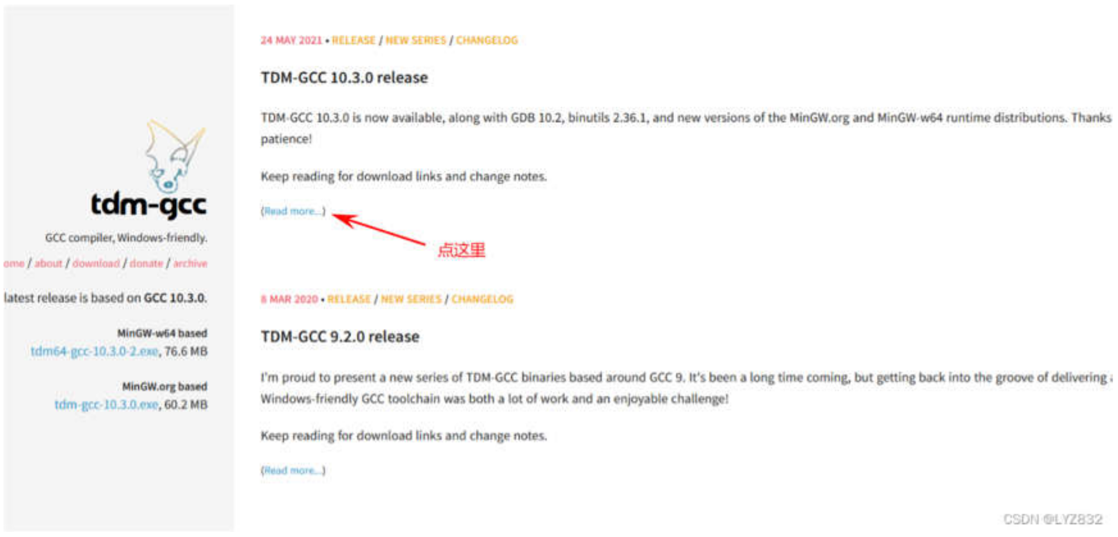
Where:
sourceVCFFile is the path to the large VCF file that serves as the data source.
targetVCFFile is the path to the generated smaller VCF file.
markersNumber is the number of markers to be extracted from the large file. If its value is 0, all markers will be retained.
samplesNumber is the number of samples to be extracted from the large file. If its value is 0, all samples will be retained.
For example, to use this tool to extract 10 markers and 5 samples from a large VCF file named ALL.chr22.20Markers.10Samples.vcf.gz in the "testFile" folder, and generate a new VCF file named ALL.chr22.10Markers.5Samples.vcf.gz, the command is as follows (assuming the current working directory is the tools folder):
./vCM ../testFile/ALL.chr22.20Markers.10Samples.vcf.gz ../testFile/ALL.chr22.10Markers.5Samples.vcf.gz 10 5

（For windows is same as Linux）
(4) readVcfTest
The source code for this tool is readVcfTest.c, located in the tools folder. Its purpose is to test the speed of reading VCF files using the vcflib library and measure the associated resource usage. It performs read operations on the VCF file without performing any further operations or modifications.
(5) readvcf.c
The tool for testing the speed of reading VCF files using htslib and measuring resource usage is available in the tools folder. It is designed to perform read operations on VCF files using htslib without any additional operations or modifications. This allows you to assess the speed and resource utilization of htslib when reading VCF files.
(6) cmWatcher.sh
You can use the tools or scripts provided by the respective operating systems to monitor the CPU usage and memory usage of a specific process and output the results to a designated text document.

Appendix 1. Set up a Windows compilation platform

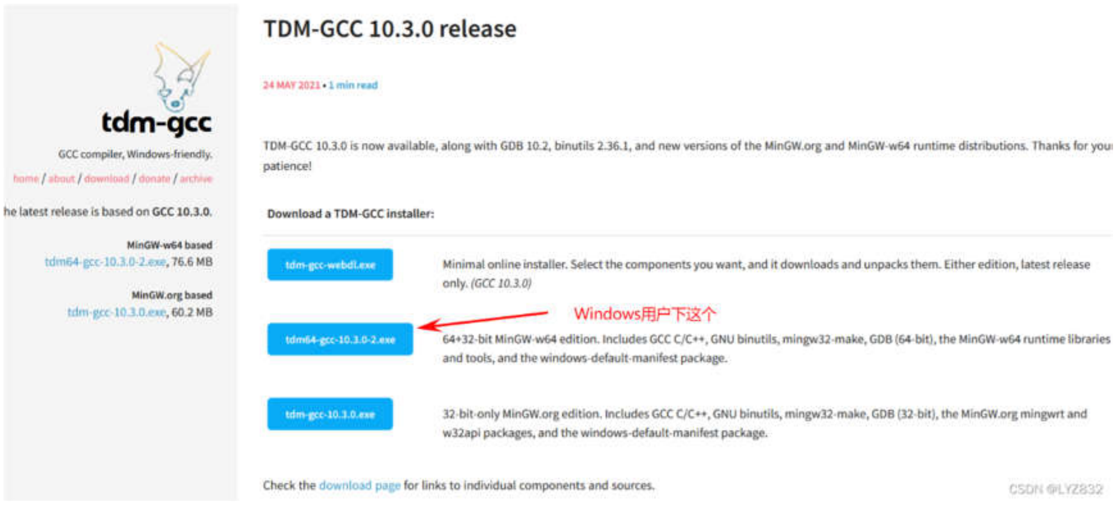
Install tdm64-gcc under Windows

(1) tdm64-gcc package
Access（https://jmeubank.github.io/tdm-gcc/）download


tdm-gcc
GCC compiler, Windows-friendly.
ome / about / download / donate / archive
latest release is based on GCC 10.3.0.
MinGW-w64 based
tdm64-gcc-10.3.0-2.exe, 76.6 MB
MinGW.org based
tdm-gcc-10.3.0.exe, 60.2 MB

24 MAY 2021 • RELEASE / NEW SERIES / CHANGELOG
TDM-GCC 10.3.0 release
TDM-GCC 10.3.0 is now available, along with GDB 10.2, binutils 2.36.1, and new versions of the MinGW.org and MinGW-w64 runtime distributions. Thanks
patience!
Keep reading for download links and change notes.
(Read more...)
点这里

8 MAR 2020 • RELEASE / NEW SERIES / CHANGELOG
TDM-GCC 9.2.0 release
I'm proud to present a new series of TDM-GCC binaries based around GCC 9. It's been a long time coming, but getting back into the groove of delivering
Windows-friendly GCC toolchain was both a lot of work and an enjoyable challenge!
Keep reading for download links and change notes.
(Read more...)


tdm-gcc
GCC compiler, Windows-friendly.
home / about / download / donate / archive
he latest release is based on GCC 10.3.0.
MinGW-w64 based
tdm64-gcc-10.3.0-2.exe, 76.6 MB
MinGW.org based
tdm-gcc-10.3.0.exe, 60.2 MB

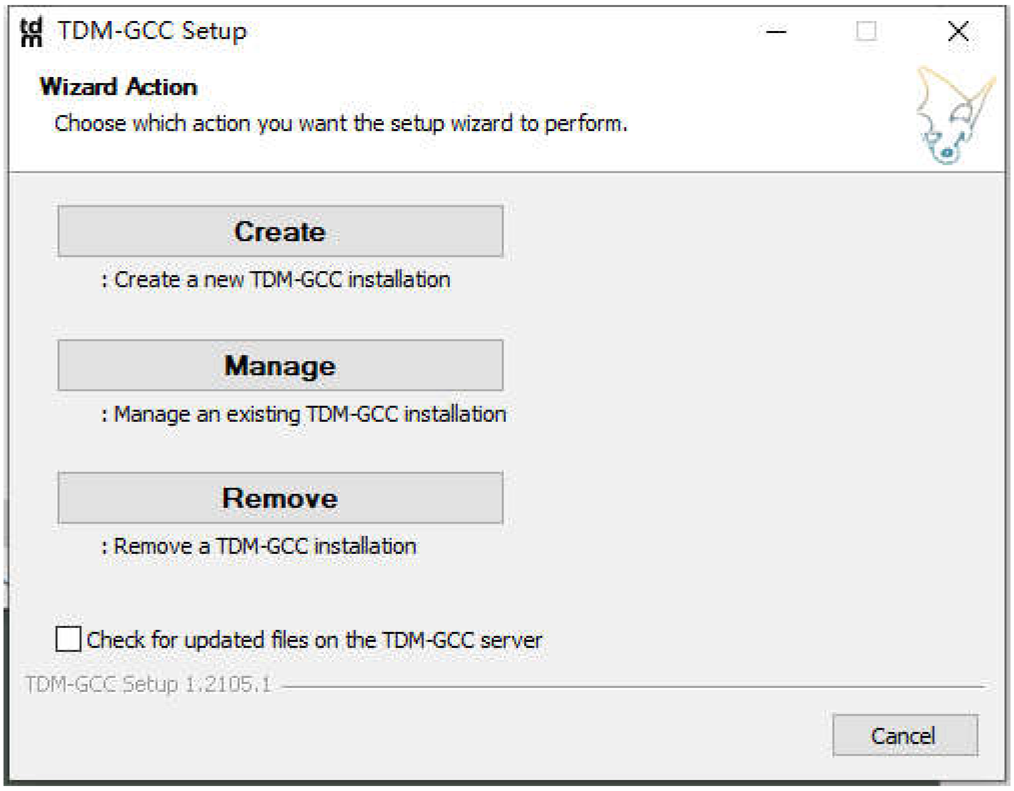
TDM-GCC 10.3.0 release
24 MAY 2021 • 1 min read
TDM-GCC 10.3.0 is now available, along with GDB 10.2, binutils 2.36.1, and new versions of the MinGW.org and MinGW-w64 runtime distributions. Thanks for you
patience!
Download a TDM-GCC installer:

tdm-gcc-webdl.exe	Minimal online installer. Select the components you want, and it downloads and unpacks them. Either edition, latest release only. *(GCC 10.3.0)*
tdm64-gcc-10.3.0-2.exe	64+32-bit MinGW-w64 edition. Includes GCC C/C++, GNU binutils, mingw32-make, GDB (64-bit), the MinGW-w64 runtime libraries and tools, and the windows-default-manifest package.
tdm-gcc-10.3.0.exe	32-bit-only MinGW.org edition. Includes GCC C/C++, GNU binutils, mingw32-make, GDB (32-bit), the MinGW.org mingwrt and w32api packages, and the windows-default-manifest package.

Windows用户下这个

Check the download page for links to individual components and sources.

Or get tdm64-gcc-10.3.0-2.exe from the folder “win_supportPackage”.
(2) Install tdm64-gcc


TDM-GCC Setup

Wizard Action

Choose which action you want the setup wizard to perform.

Create
: Create a new TDM-GCC installation

Manage
: Manage an existing TDM-GCC installation

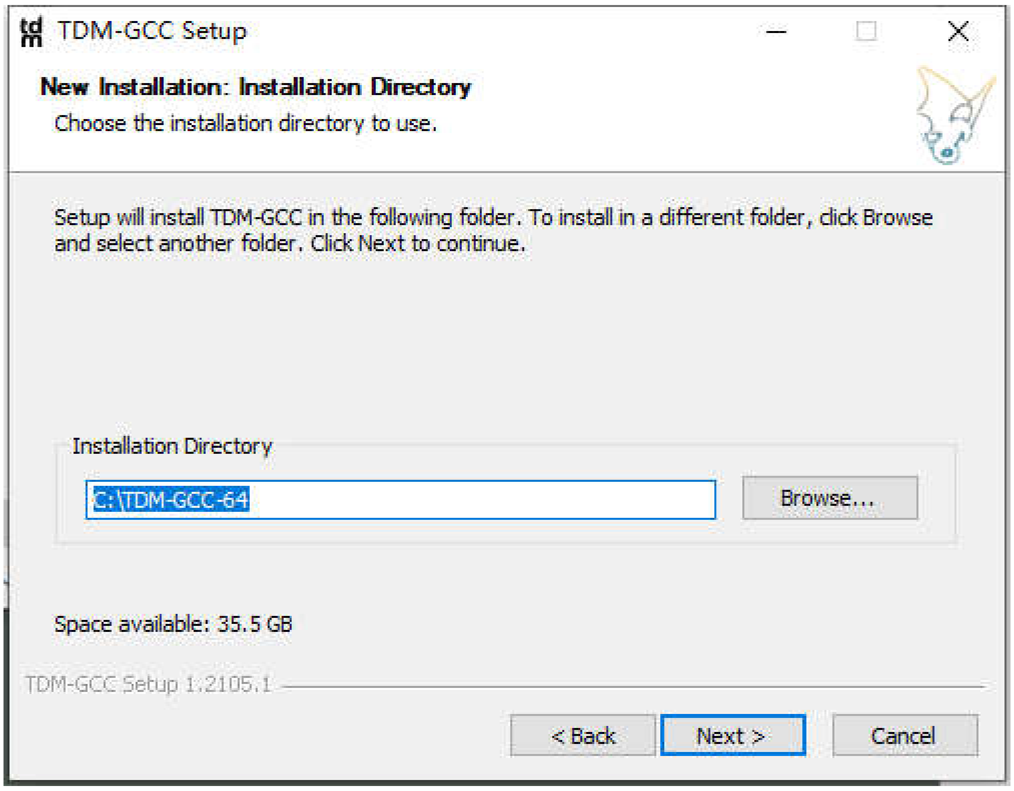
Remove
: Remove a TDM-GCC installation

☐ Check for updated files on the TDM-GCC server

TDM-GCC Setup 1.2105.1

Cancel

Cancel 'check for updated files on the TDM-GCC server ' option， then click Create。


TDM-GCC Setup

New Installation: Installation Directory

Choose the installation directory to use.

Setup will install TDM-GCC in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Installation Directory

C:\TDM-GCC-64

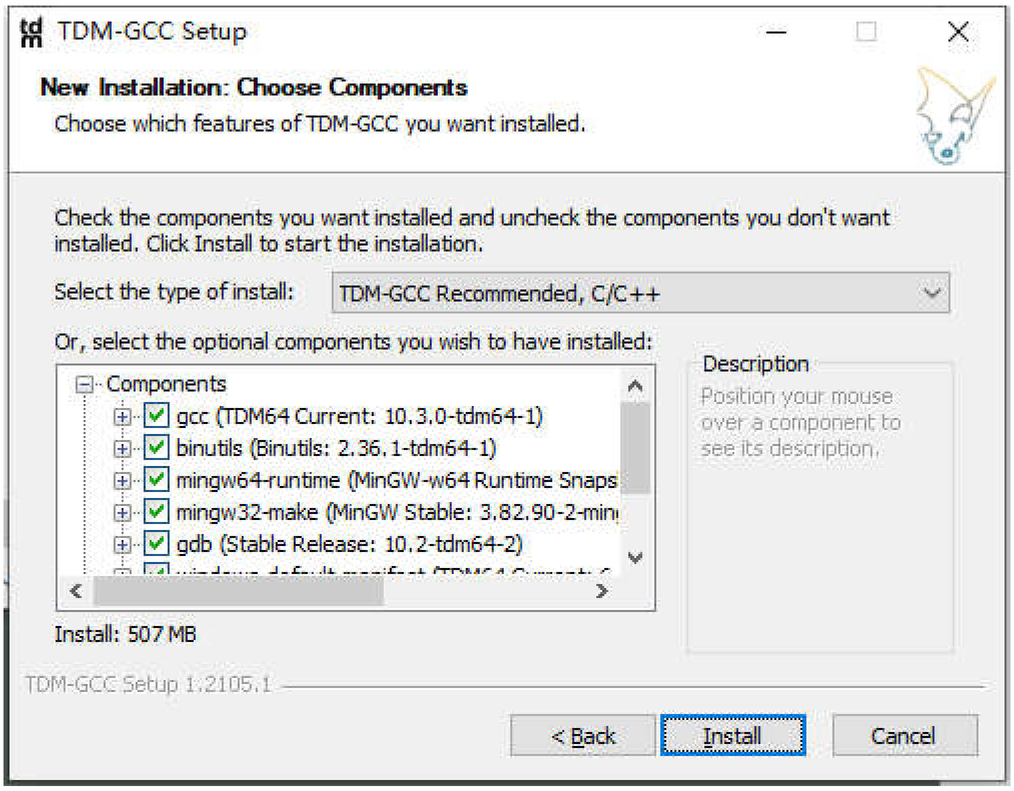
Browse...

Space available: 35.5 GB

TDM-GCC Setup 1.2105.1

< Back　Next >　Cancel

Choose a pathway without special character, then click Next.



TDM-GCC Setup

New Installation: Choose Components

Choose which features of TDM-GCC you want installed.

Check the components you want installed and uncheck the components you don't want installed. Click Install to start the installation.

Select the type of install: TDM-GCC Recommended, C/C++

Or, select the optional components you wish to have installed:

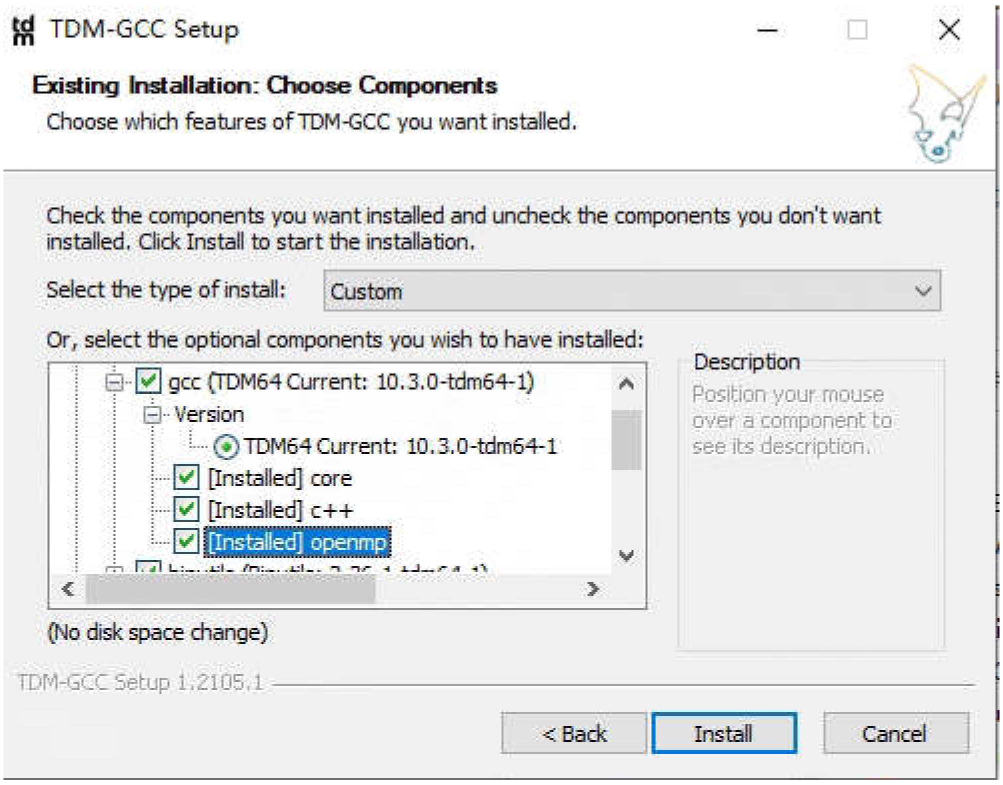
- Components
  - [✓] gcc (TDM64 Current: 10.3.0-tdm64-1)
  - [✓] binutils (Binutils: 2.36.1-tdm64-1)
  - [✓] mingw64-runtime (MinGW-w64 Runtime Snaps
  - [✓] mingw32-make (MinGW Stable: 3.82.90-2-min
  - [✓] gdb (Stable Release: 10.2-tdm64-2)

Description: Position your mouse over a component to see its description.

Install: 507 MB

TDM-GCC Setup 1.2105.1

Select all options. Pay attention to the first option 'gcc' and make sure it includes OpenMP. It must be selected. See the image below:



TDM-GCC Setup

Existing Installation: Choose Components

Choose which features of TDM-GCC you want installed.

Check the components you want installed and uncheck the components you don't want installed. Click Install to start the installation.

Select the type of install: Custom

Or, select the optional components you wish to have installed:

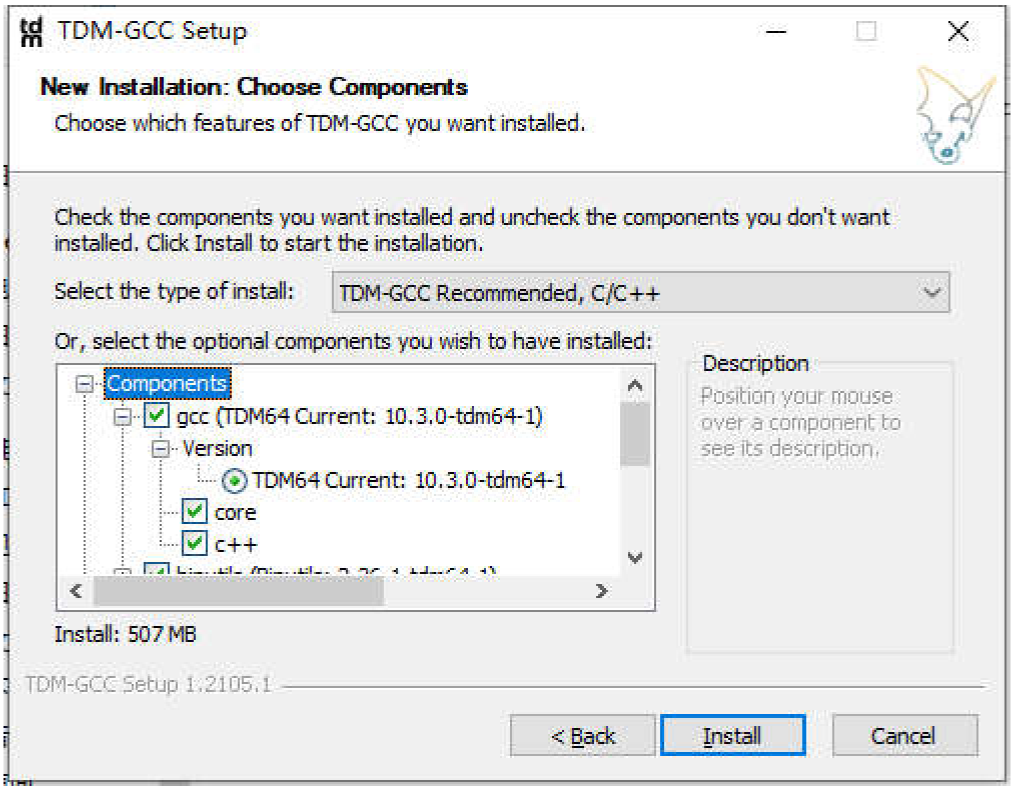
- [✓] gcc (TDM64 Current: 10.3.0-tdm64-1)
  - Version
    - ● TDM64 Current: 10.3.0-tdm64-1
  - [✓] [Installed] core
  - [✓] [Installed] c++
  - [✓] [Installed] openmp

Description: Position your mouse over a component to see its description.

(No disk space change)

TDM-GCC Setup 1.2105.1

If OpenMP is not available in the first option, as shown in the image below:


TDM-GCC Setup

New Installation: Choose Components

Choose which features of TDM-GCC you want installed.

Check the components you want installed and uncheck the components you don't want installed. Click Install to start the installation.

Select the type of install: TDM-GCC Recommended, C/C++

Or, select the optional components you wish to have installed:

- Components
  - [✓] gcc (TDM64 Current: 10.3.0-tdm64-1)
    - Version
      - ● TDM64 Current: 10.3.0-tdm64-1
    - [✓] core
    - [✓] c++

Description

Position your mouse over a component to see its description.

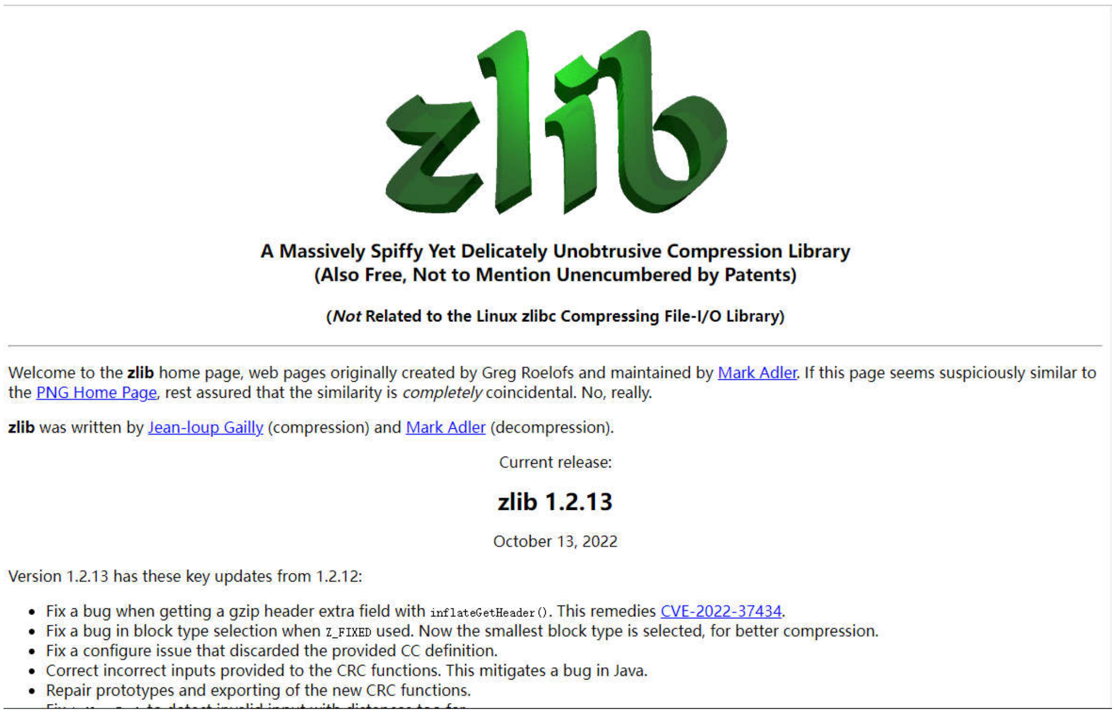
Install: 507 MB

TDM-GCC Setup 1.2105.1

After the installation is complete, you need to double-click on the installation package again to enter the installation wizard and choose 'Manage' for the installed TDM-GCC. Then, make sure to add OpenMP during the management process. This step is necessary to enable proper compilation of vcflib. Finally, click 'Install' to complete the installation.

Install zlib under Windows

(1) zlib installation package
go to（https://www.zlib.net/）download



A Massively Spiffy Yet Delicately Unobtrusive Compression Library
(Also Free, Not to Mention Unencumbered by Patents)

(*Not* Related to the Linux zlibc Compressing File-I/O Library)

Welcome to the **zlib** home page, web pages originally created by Greg Roelofs and maintained by Mark Adler. If this page seems suspiciously similar to the PNG Home Page, rest assured that the similarity is *completely* coincidental. No, really.

zlib was written by Jean-loup Gailly (compression) and Mark Adler (decompression).

Current release:

zlib 1.2.13

October 13, 2022

Version 1.2.13 has these key updates from 1.2.12:

- Fix a bug when getting a gzip header extra field with `inflateGetHeader()`. This remedies CVE-2022-37434.
- Fix a bug in block type selection when `Z_FIXED` used. Now the smallest block type is selected, for better compression.
- Fix a configure issue that discarded the provided CC definition.
- Correct incorrect inputs provided to the CRC functions. This mitigates a bug in Java.
- Repair prototypes and exporting of the new CRC functions.

Alternatively, in the “win_supportPackage” that I provided, there is a pre-downloaded tar package named 'zlib-1.2.13.tar.gz'zlib.
(2) Install zlib
In Windows, open the command prompt (cmd). Extract the zlib installation package to any directory. In this example, I'll assume you've placed the zlib package in 'D:\tmp' and will perform the extraction locally. Enter the following command.

```
d:
cd tmp
tar -xzvf zlib-1.2.13.tar.gz
```

Go into the extracted zlib folder. Copy the file 'win32\Makefile.gcc' from the zlib folder to the root of the zlib directory. Enter the following command.

```
cd zlib-1.2.13
copy win32\Makefile.gcc makefile.gcc
```

To compile zlib, enter the following command:

```
Mingw32-make -f makefile.gcc
```

After the compilation is complete, copy the compiled zconf.h and zlib.h files to the include folder in the tdm64-gcc installation directory. Also, copy the compiled libz.a file to the lib folder in the tdm64-gcc installation directory.