Artifact Instructions for MARTINI: The Little Match and Replace Tool

Alister Johnson[1*], Camille Coti[2], Allen D. Malony[3], and Johannes Doerfert[4]

[1] University of Oregon, Eugene, OR, USA
ajohnson@cs.uoregon.edu
[2] Université du Québec à Montréal, Montréal, QC, Canada
coti.camille@uqam.ca
[3] University of Oregon, Eugene, OR, USA
malony@cs.uoregon.edu
[4] Argonne National Laboratory, Lemont, IL, USA
jdoerfert@anl.gov[0000−0001−7870−8963]

md5 of code .zip file: de0705bbcb4109022931b1c56b769df6

1 Getting Started

1.1 Required dependencies:

- CMake (version $\geq$ 3.13.4, we primarily used 3.22.1) – full instructions: https://cmake.org/install/. Short instructions:
  - Download source from https://cmake.org/download/
  - Unzip, enter `cmake-x.xx.x` directory
  - Run the following (assumes Linux):
    * `./bootstrap`
    * `make`
    * `make install`

1.2 Optional dependencies:

- Ninja build system (we primarily used version 1.10.0), https://ninja-build.org/ –
  - Download via package manager (as we did): https://github.com/ninja-build/ninja/wiki/Pre-built-Ninja-packages
  - OR download source (https://github.com/ninja-build/ninja) and build (https://github.com/ninja-build/ninja/wiki)

1.3 To build MARTINI:

1. Install CMake and Ninja (recommended for quicker build of LLVM, but not required)
2. Clone our fork of the LLVM monorepo OR unzip copy on artifact:

[*] Work done while at Argonne National Laboratory.

   - `git clone https://github.com/ajohnson-uoregon/llvm-project.git`
3. Checkout `europar22-artifact` branch:
   - `git checkout europar22-artifact`
4. Follow instructions to build LLVM, including `clang` and `clang-tools-extra` projects. Full instructions: https://llvm.org/docs/GettingStarted.html. Short instructions:
   - Enter `llvm-project` directory
   - `mkdir build`
   - `cd build`
   - For Ninja (recommended):
     - `cmake -G Ninja \`
       `-DLLVM_TARGETS_TO_BUILD="X86;NVPTX;AMDGPU" \`
       `-DCMAKE_INSTALL_PREFIX="<install dir>" \`
       `-DCMAKE_BUILD_TYPE=RelWithDebInfo \`
       `-DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra" ../llvm`
     - `ninja -j <N> install`
   - For Unix make:
     - `cmake -G "Unix Makefiles" \`
       `-DLLVM_TARGETS_TO_BUILD="X86;NVPTX;AMDGPU" \`
       `-DCMAKE_INSTALL_PREFIX="<install dir>" \`
       `-DCMAKE_BUILD_TYPE=RelWithDebInfo \`
       `-DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra" ../llvm`
     - `cmake --build . -j <N> --target install`
5. MARTINI/`clang-rewrite` executable will be in the install directory chosen when building LLVM.

1.4 Notes on Build

- The majority of our contributions are in `llvm-project/clang-tools-extra/clang-rewrite`.
- LLVM/Clang is a *very* large project, and will take several hours to build even when done in parallel.
- Parallel builds are *highly* recommended, but will use a large amount of memory when linking the final libraries and executables. The last approx. 500 targets may need to be built in serial if your machine has ≤24GB of memory.

2 Instructions to Reproduce Results

2.1 modernize-use-nullptr (Introduction)

In `llvm-project/clang-tools-extra/clang-rewrite/tests/modernize-use-nullptr`, run:

```
clang-rewrite --inst-file=spec.cpp test.cpp --
```

The file spec.cpp contains the matchers and replacers used to emulate modernize-use-nullptr's functionality. The file test.cpp contains the code to be rewritten and the expected result.

The above command should only take a few seconds and produce a file named test.test_final.cpp with the rewritten code (and expected result for comparison). This example corresponds to the one presented in the paper's Introduction.

2.2 Instrumentation (First Case Study)

In `llvm-project/clang-tools-extra/clang-rewrite/tests/inst`, run:

```
clang-rewrite --inst-file=inst.cpp test.cpp -- --std=c++20
```

The file inst.cpp contains the matchers and replacers used to instrument single-argument function calls and lambdas. The file test.cpp contains the code to be rewritten and the expected result.

The above command should only take a few seconds and produce a file named test.test_final.cpp with the rewritten code (and expected result for comparison). This example corresponds to the one presented in Section 3, the first case study on instrumentation.[1]

2.3 HIPIFY (Second Case Study)

Dependencies:

- A machine with an AMD GPU and associated drivers and compilers. Our machine had:
  - two 14-core, hyperthreaded Intel Xeon(R) E5-2680v4 CPUs running at 2.40GHz
  - 128 GB of RAM
  - two AMD Instinct MI100 GPUs
  - `hipcc` 4.4.21432-f9dccde4 based on AMD Clang 13.0.0 and ROCm 4.5.2
- `hipify-perl` (we had the same version as `hipcc` above)
- `hipify-clang` (we used git hash 61241a4 and compiled with gcc 9.3.0)

Both `hipify-perl` and `hipify-clang` can be obtained from https://github.com/ROCm-Developer-Tools/HIPIFY.

Building AMD's HIPIFY: Short version of the instructions at the above link for building `hipify-clang` and `hipify-perl`:

1. Create a directory for HIPIFY and enter it:
   - `mkdir HIPIFY`

[1] There is a small bug in the rewriting infrastructure for the artifact version of `clang-rewrite` that sometimes leaves an extra character behind in the output for this example. As of submission, the rewriting infrastructure is undergoing major improvements to make it less bug-prone going forward.

   - `cd HIPIFY`
2. Clone AMD's repository and checkout the `rocm-4.5.2` branch (this is the version we used):
   - `git clone https://github.com/ROCm-Developer-Tools/HIPIFY.git`
   - `git checkout rocm-4.5.2`
3. In top level `HIPIFY` directory (*not* the `hipify` directory created by the clone) make directories for the build:
   - `mkdir build dist`
   - `cd build`
4. In the `build` directory, run cmake and make (note: you may have to install libtinfo-dev with your package manager):
   - ```
     cmake \
     -DCMAKE_INSTALL_PREFIX=../dist \
     -DCMAKE_BUILD_TYPE=Release \
     ../hipify
     ```
   - `make -j install`
5. The `hipify-clang` executable should now be in `HIPIFY/dist`. To generate `hipify-perl`:
   - `cd ../dist`
   - `./hipify-clang --perl`
6. `hipify-clang` and `hipify-perl` should now both be in the `dist` directory.

Running the tests:
In `llvm-project/clang-tools-extra/clang-rewrite/tests/hipify`:

- Unzip mini-nbody.zip, enter `mini-nbody-master/cuda` directory
- Produce `hipify-perl` and/or `hipify-clang` versions of each test in `mini-nbody-master/cuda`, following directions here: https://github.com/ROCm-Developer-Tools/HIPIFY. (Note: You may have to manually delete the SHMOO `#ifdef`s, some versions of HIPIFY don't handle them well.) Example commands:
  - `hipify-clang nbody-orig.cu -- -I..`
  - `perl hipify-perl nbody-orig.cu > nbody-orig.hip`
- Produce MARTINI/`clang-rewrite` versions of the tests in `mini-nbody-master/cuda` by running, e.g.,
  - `clang-rewrite --inst-file=../../hipify.cpp nbody-orig.cu -- -I.. -I../.. -x cuda --std=c++20`
- MARTINI/`clang-rewrite` versions with two times the number of threads at kernel launch can be produced by replacing `hipify.cpp` in the above with `hipify-x2-threads.cpp`.
- Compile each test, e.g. with `hipcc nbody-orig.cu.hip -I.. -o nbody-orig`.
- Run each test (e.g., `./nbody-orig`). Output should look something like this:
     ```

```
Iteration 1: 0.499 seconds
Iteration 2: 0.191 seconds
Iteration 3: 0.191 seconds
Iteration 4: 0.191 seconds
Iteration 5: 0.191 seconds
Iteration 6: 0.191 seconds
Iteration 7: 0.191 seconds
Iteration 8: 0.191 seconds
Iteration 9: 0.203 seconds
Iteration 10: 0.192 seconds
300000 Bodies: average 467.571 Billion Interactions / second
```

Interpreting results:
Times in Table 1 in the paper are reported in ms. The average and standard deviation time in the table were calculated over the 10 iterations that each test runs. The first two sets of results in the table (columns under MARTINI-HIPIFY labeled "Unmodified" and "#Threads x2") were found using `clang-rewrite` with `hipify.cpp` and `hipify-x2-threads.cpp`, respectively. We expect performance from `hipify.cpp` to always match performance from `hipify-clang` and `hipify-perl`, since they should produce nearly identical code. We expect performance from `hipify-x2-threads.cpp` to be somewhat faster on large problem sizes for all tests except nbody-orig (which is unoptimized and not expected to parallelize well). How much faster will depend on the GPU.

2.4 Notes on Running `clang-rewrite`

- The `--inst-file` argument is a holdover from when MARTINI/`clang-rewrite` was envisioned as purely an instrumentation tool, and will be changed to `--spec-file`.
- The `--` at/near the end of the `clang-rewrite` (and `hipify-clang`) commands *is* necessary; it is how Clang's libtooling infrastructure detects which commands are to be passed directly to the compiler, in lieu of reading from a compilation database.
- As this is a prototype still in very active development, there is a decent amount of debugging information dumped to `stdout` to assist in debugging. If you run into any errors, please send us the specification file, code file, and any output from `clang-rewrite`.
- Some of the example matchers and replacers in the paper were edited for brevity/space, as mentioned in the paper. The matchers and replacers in the artifact use the full syntax `clang-rewrite` currently expects.