A. Artifact Appendix

A.1 Abstract

This artifact provides the source code that implements our function merging optimization as well as the other optimizations required for our evaluation. Our optimization is implemented on top of LLVM v7. We also provide the source code for all benchmarks along with scripts required to reproduce the results presented in the paper. To validate the results build our version of LLVM with the provided scripts, run the benchmarks and, finally, the plotting script to reproduce the main results in the paper.

A.2 Artifact check-list

- **Program:** LLVM and Clang, the C/C++ frontend for LLVM; the C/C++ SPEC CPU2006 benchmark suite.
- **Compilation:** With provided scripts.
- **Data set:** Provided with the corresponding benchmarks.
- **Run-time environment:** Linux.
- **Hardware:** Intel architecture.
- **Output:** Raw data in CSV files and plots as PDFs.
- **How much disk space required (approximately)?:** Up to 5 GiB.
- **Publicly available?:** Yes.
- **Workflow frameworks used?:** Download and unzip; build software; run benchmarking scripts; compare output results with the expected plots provided.

A.3 Description

A.3.1 How delivered

The artifact is publicly available. We provide two options to reproduce our experiments:

- Download the source code and benchmark suite, building everything locally on your own machine.

  `http://bit.ly/cgo19fmsa`

  `http://bit.ly/cgo19fmsa-benchmark`
- Download our pre-packaged VirtualBox image with the LLVM built and ready to run the benchmarking scripts.

  `http://bit.ly/cgo19fmsa-vm`

  The password for this system is `cgo19fmsa`, the same as the username.

The main source file that implements our optimization can be found in the path:

`CGO19FMSA/llvm/lib/Transforms/IPO/FunctionMerging.cpp`

A.3.2 Hardware dependencies

The experiments described by this artifact were executed on an Intel machine with Intel Core i7-4770 CPU at 3.40 GHz, and 16 GiB of RAM.

A.3.3 Software dependencies

In this section, we describe the softwares and packages that must be installed in order to build the LLVM compiler, the benchmark suite, and produce the plots with the results.

The experiments described by this artifact were executed on a machine with the operating system openSUSE Leap 42.2.

Below, we list all Linux and Python packages that must be installed. We also specify the exact version that we have used in our experiments.

- GCC for both C and C++ (`gcc` , `g++`)

  `gcc-4.8-9.61.x86_64`

  `gcc-c++-4.8-9.61.x86_64`

  `binutils-2.29.1-9.6.1.x86_64`
- GCC's 32-bits runtime (`gcc-multilib`, `g++-multilib`)

  `gcc-32bit-4.8-9.61.x86_64`

  `gcc-c++-32bit-4.8-9.61.x86_64`
- CMake build system (`cmake`)

  `cmake-3.5.2-1.2.x86_64`
- Python 2.7+ (`python`)

  `python-2.7.13-25.3.1.x86_64`
- Python's TkInter (`python-tk`)

  `python-tk-2.7.13-25.3.1.x86_64`
- Python's pip (`python-pip`)

  `python-pip-7.1.2-2.4.noarch`
- Python's NumPy (`numpy`)

  `numpy 1.14.2`
- Python's Matplotlib (`matplotlib`)

  `matplotlib 2.1.0`
- Python's Seaborn (`seaborn`)

  `seaborn 0.9.0`

We provide a script, called `setup.sh`, which automatically installs all the necessary dependencies. This script uses the `apt` tool and is the only one that requires *sudo* privileges.

A.3.4 Data sets

Datasets are provided as part of the artifact with the benchmark suite.

A.4 Installation

Download both the source code (`http://bit.ly/cgo19fmsa`) and the benchmark suite (`http://bit.ly/cgo19fmsa-benchmark`). The source code has a root directory called CGO19FMSA. Unzip all the content comprising the benchmark suite inside the CGO19FMSA root directory. At this point, your CGO19FMSA directory should contain:

```
./CGO19FMSA/
├── build-all.sh
├── config.sh
├── data
├── expected
├── llvm
├── myplots.py
├── plot-code-size.py
├── plot-compilation-time.py
├── plot-execution-time.py
├── run-all.sh
├── setup.sh
└── spec2006
```

In order to install all dependencies described above, run the `setup.sh` script with *sudo* privileges. That is, assuming that you are in the CGO19FMSA directory, run the following command:

```
sudo sh setup.sh
```

Once the dependencies have been installed, run the `build-all.sh` script to build our version of LLVM and Clang, which include our function merging optimization as well as both the state-of-the-art optimization and LLVM's identical function merging optimization. Again, assuming that you are in the CGO19FMSA directory, run the following command:

```
sh build-all.sh
```

This process might take a few hours, depending on your machine settings. After completion, a `build` directory is created inside the CGO19FMSA root directory.

This two scripts set up all the environment necessary to run all our experiments.

Our pre-packaged VirtualBox image (`http://bit.ly/cgo19fmsa-vm`) has all this environment setup and ready to run the experiments, as described in the next section.

A.5 Experiment workflow

To run our experiments all you need to do is to execute the `run-all.sh` script with the following command:

```
sh run-all.sh
```

This script automates the whole experiment workflow. At the end, you should have all the expected plots, as well as the raw data as CSV files, inside the `results` directory, which is also created inside the CGO19FMSA root directory.

The automated process may take several hours since it involves running all following steps for all the SPEC benchmarks:

- Code-size measurement (Figure 10 in the paper):
  - Running the oracle optimization.
  - Running the state-of-the-art and baselines optimizations.
  - Running our optimization with all three exploration thresholds.
- Compilation-time measurement (Figure 11 in the paper):
  - Run all optimizations again, except for the oracle, multiple times in order to have a measurement with statistical significance.
- Execution-time measurement (Figure 13 in the paper):
  - Run, also multiple times, all compiled versions of the SPEC benchmarks with their reference inputs.

A.6 Evaluation and expected result

After executing the automated process described above, the `results` directory should have the following content:

```
results/
└── spec2006
    ├── code-size-reduction.pdf
    ├── compilation-time.pdf
    ├── execution-time.pdf
    ├── n1
    │   ├── compilation.csv
    │   ├── exec.csv
    │   └── results.csv
    ├── n10
    │   ├── compilation.csv
    │   ├── exec.csv
    │   └── results.csv
    ├── n5
    │   ├── compilation.csv
    │   ├── exec.csv
    │   └── results.csv
    └── oracle
        └── results.csv
```

The main files to consider are the PDF files, which represent the plots with the main results for the final version of our paper. The three PDF files, as the name of the files suggest, contain the results regarding code-size reduction, compilation-time overhead, and the performance impact of our optimization during execution-time of the benchmarks.

This artifact represents our results for the camera ready version of the paper. We provide the `expected` results that were produced in our environment with the up-to-date version that will be used in the camera ready version of the paper (for example, it includes a benchmark that was missing in the original version of our paper). The reviewers can compare their results with the ones provided in the `expected` directory.

A.7 Notes

To reduce the overall time required to run the full set of experiments, we have set a small number of repetition, which may result in large error bars for some of the benchmarks. This effect will depend on the noise of your environment. In order to reduce the noise in the measurements, you just need to update the `REPEAT` variable in the *CGO19FMSA/run-all.sh* script, changing it to a bigger number.

Because SPEC 2006 requires private access, in the final version, we will change `http://bit.ly/cgo19fmsa-benchmark` to provide only the scripts for running the experiments. The source code in `http://bit.ly/cgo19fmsa` will remain the same.