A. Artifact Appendix

A.1 Abstract

This artifact provides the source code that implements our function merging optimization, called SalSSA, as well as the state-of-the-art optimization. Both optimizations are implemented on top of LLVM v11. 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.
- **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 10 GiB.
- **Publicly available?:** Yes.
- **Workflow frameworks used?:** Download and unzip; build software; run benchmarking scripts; confirm output results and plots.

A.3 Description

A.3.1 Delivered artifact

The artifact is publicly available (DOI: 10.6084/m9.figshare.12089217). Download the archive in the link below:
`https://figshare.com/s/df706779587ba6ead4b2`

Please, note that the link shows a warning due to the size of the file (over 100 MB), but there is a link to download the file `pldi20ae.tar.gz`.

This archive includes the modified LLVM source code, the benchmark suite, and the necessary scripts. The zipped archive contain a folder called `pldi20salssa-ae` which includes all the files for this artifact. In particular, there are two sub-directories called: `llvm-project` and `eval`. The first contains the modified LLVM source code and the second contains the SPEC 2006 benchmark suite with scripts.

The main source file that implements both the state-of-the-art and our optimization (SalSSA) can be found in the path:

```
llvm-project/llvm/lib/Transforms/IPO/FunctionMerging.cpp
```

SalSSA is enabled by setting the flag `-func-merging-salssa` when executing the LLVM optimizer. In the `FunctionMerging.cpp` source file, the SalSSA code generator is implemented by the class `SALSSACodeGen` and its helper functions. Except for the code generator, SalSSA shares most of its code base with the state-of-the-art optimization (FMSA).

A.3.2 Hardware dependencies

The experiments described by this artifact were executed on an Intel machine with Intel Core i7-4770 CPU 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`

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. This is a small and easily readable script, if the user prefers to run each install command separately.

A.4 Installation

After downloading the archive from the provided link and unzipping the `pldi20salssa-ae` directory, install the dependencies described above. In order to install all dependencies, run the `setup.sh` script with *sudo* privileges. That is, assuming that you are in the `pldi20salssa-ae` 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 the state-of-the-art optimization. Still assuming that you are in the `pldi20salssa-ae` directory, run the following command:

```
sh build-all.sh [Number of Jobs]
```

This process might take a few hours, depending on your machine settings. In order to speed up the process, the user can run the build script with a higher number of parallel jobs. A larger number of jobs also implies a larger amount of memory usage. After completion, a `build` folder is created in the `pldi20salssa-ae` root directory. The resulting `pldi20salssa-ae/build/bin` directory should include the built programs: `clang`, `opt`, and `llvm-link`.

A.5 Experiment workflow

To run our experiments, inside the `pldi20salssa-ae/eval` directory, you only need to run the `run-all.sh` script with the following command:

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

This script automates the whole experiment workflow, producing the results for both code-size and compilation time. At the end, you should have the final plots, as well as the raw data as CSV files, inside the `results` directory, which will be created inside the `pldi20salssa-ae/eval` directory. The `run-all.sh` script includes a `COALESCING` (`true/false`) variable that specifies whether it should enable phi-node coalescing.

The automated process may take up to a few hours to run the full code-size evaluation. Alternatively, we also provide a script for a quick check. The `run-sample.sh` script runs the same workflow on a small subset of the benchmarks. To run this quick check, run the following command from inside the `pldi20salssa-ae/eval` directory:

```
sh run-sample.sh
```

This script should take about 5 minutes or less to run and produce a `results` directory similar to that from the full workflow.

A.6 Evaluation and expected result

After executing the automated process described above, the `results` directory should include two PDF files called `code-size-reduction.pdf` and `compilation-time.pdf`. These files contain key results from our paper, showing how SalSSA improves over the state-of-the-art (FMSA).

A.7 Notes

Because SPEC 2006 requires private access, in the final version, we have changed the archive to provide only the scripts for running the experiments, without the actual source code of the benchmark suite. The modified LLVM source code remains publicly available.