Artifact for the paper *AUTOMAP: Inferring Rank-Polymorphic Function Applications with Integer Linear Programming*

Introduction

This artifact reproduces the quantitative evaluation discussed in Section 9 of the paper, in particular reproducing Fig. 12 and Fig. 13.

We perform a comparative study on a set of pairs of similar benchmark programs. For each pair, one of the programs is the "original", and the other has been rewritten to take advantage of the "AUTOMAP" feature discussed in the paper.

Hardware dependencies

None, although the Docker image uses x86-64 binaries, and so the system must be capable of running such an image. The memory requirements are modest (less than 1GiB).

Getting started

The artifact takes the form of a Docker image `607.tar.gz`. You can load it into Docker with this command:

```
$ docker load -i 607.tar.gz
```

(Depending on your system configuration, this may or may not require root access.)

You can then run the Docker image with this command:

```
$ docker run -it 607:latest
```

Step-by-Step instructions

Running `make` will reproduce the quantitative evaluation discussed in Section 9 of the paper. This takes about 20 minutes on a modern computer. The results are printed on the terminal as they come in, and are also stored as files in the `results/` directory. A `data/` directory containing raw (unprocessed) data is also constructed, but can be ignored. Its contents are described further down for the benefit of future users who want to more deeply investigate the results. Finally, the artifact reproduces Fig. 12 and Fig. 13 from the paper.

- Fig. 12 is available as `reports/fig12.pdf` and `reports/fig12.txt`, with the latter also printed to the terminal.
- Fig. 13 is available as `reports/fig13.txt`, and is also printed to the terminal. The reported slowdown can vary from machine to machine, but the other metrics should match the paper exactly.

This completes the evaluation of the functionality as far as concerns reproducing the quantitative claims in the paper.

Retrieving the figures from the Docker container

To copy a figure from the container for viewing, first obtain the container ID by running

```
$ docker ps
```

You can then copy the figure using `docker cp`:

```
$ docker cp <container ID>:<path to figure in the container> <destination>
```

Interactive use

If desired, AUTOMAP can be tried by starting a REPL with

```
$ futhark-automap repl
```

and entering valid expressions. Examples:

```
> [1,2,3] + 2
[3, 4, 5]
>  [1,2,3] * transpose (rep [4,5,6])
[[4, 8, 12],
 [5, 10, 15],
 [6, 12, 18]]
```

Reusability Guide

The artifact is reasonably straightforward to modify in two ways: modifying which benchmark programs are considered, and performing analysis on the raw data.

Changing benchmark programs

In `data.sh`, modify the shell function `programs`. This function produces paths to `.fut` programs that are relative to the directory `futhark-benchmarks-original` or `futhark-benchmarks-automap`. It is important that a version of the program exists in both of these directory trees (corresponding to a version of the program without and with use of AUTOMAP).

Raw data files

After running the artifact, the `data/` directory contains raw data. This data is processed to reproduce Fig. 12 and Fig. 13, but can also be subjected to other investigation. The following data is produced.

- `data/ilps`: contains a pair of files for each benchmark program `foo.fut`: an `.ilps` file (containing a table of the ILP size for each function in the program), and a `.log` file containing the raw (unprocessed) compiler logging output.
- `data/ilp_table`: a table of the sizes of all ILP problems, keyed by the (internal) name of the function that gave rise to them.
- `data/ilp_sizes`: a sorted list of the sizes of all ILP problems produced during type checking the benchmark programs.
- `data/ilp_stats`: various statistical measures derived from `data/ilp_sizes`, including mean and median.
- `data/ilp_largest`: the size (in number of constraints) of the largest ILP problem encountered.
- `data/maps_automap`: the total number of `map`s used in the AUTOMAP-enabled benchmark programs.
- `data/maps_original`: the total number of `map`s used in the original benchmark programs.
- `data/maps.txt`: contains a line for each benchmark program, comprising the name of the program, the number of `map`s in the original program, and the number of `map`s in the AUTOMAPped program.

Running outside Docker

The following system requirements are satisfied by the Docker image, and are only listed for the sake of completeness, in particular if you want to run it outside of Docker.

PATH must contain two compiler binaries `futhark-original` and `futhark-automap`, corresponding to the unmodified and AUTOMAP-enabled Futhark compiler.

The following tools must also be available:

- scc
- hyperfine
- gnuplot
- bc
- awk
- A handful of standard command like tools like `grep`, `find`, `wc`, etc.

Manifest

This section describes every top-level file and directory in the artifact and its purpose.

- `data.sh`: The main data analysis script, invoked by `Makefile`.
- `findilps.awk`: An Awk script that extracts ILP programs from compiler logs.
- `futhark-benchmarks-automap`: The Futhark benchmark suite modified to take advantage of `automap`.
- `futhark-benchmarks-original`: The unmodified Futhark benchmark suite.
- `Makefile`: A simplistic Makefile that simply runs `data.sh`.

More files and directories are created as part of the artifact as discussed above.