Artifact of "Contextual Dispatch for Function Specialization" [OOPSLA'20]

Olivier Flückiger Guido Chari Jan Ječmen
Ming-Ho Yee Jakob Hain Jan Vitek

This is the artifact to accompany our OOPSLA 2020 submission on "Contextual Dispatch for Function Specialization".

The artifact consists of a virtual machine for the R language, called Ř, a suite of benchmarks written in R, as well as an R script to interpret and plot the results. The Ř VM is included in source format with build instruction, and it is distributed as OCI images available through our container registry and attached as `images-502.tgz`.

1 Getting Started

The following instructions were tested on Ubuntu Bionic.

The results in our paper are based on commit bc1933d. To fetch the appropriate image and start an Ř build at that version, it suffices to run the following command using your favorite OCI compliant container runtime, such as docker or podman:

```
export CR=docker
export VS=bc1933dde2673bf830f4505bb2483cd1fdd282ab
export RG=registry.gitlab.com/rirvm/rir_mirror/splash20
$CR run -it $RG:$VS /opt/rir/build/release/bin/R
```

The image that you receive is based on the Dockerfile at the root of our repository. It should have a SHA2-256 of

```
c7f92d3abc20e65f8e3ff50b3d8f0967a3c53017863ab9dad77bd447477698c9
```

which you can verify with:

```
$CR inspect --format='{{index .RepoDigests 0}}' $RG:$VS
```

To get a feeling for the contextual dispatching presented in this paper, you can copy paste the following program into the Ř REPL:

```
f <- function(a,b) a+b

# Trigger ast to BC compilation
f(1,1); f(1,1)

# Inspect baseline version:
rir.disassemble(f)

# Trigger specialization to integer
f(1L,1L); f(1L,1L); f(1L, 1L)

# The function has now two versions. The second version is compiled
# under a context (Assumptions) that includes SimpleInt0,SimpleInt1,
# indicating the first two arguments being scalar integers.
rir.disassemble(f)
```

The newest version of Ř can be run using:

```
$CR run -it registry.gitlab.com/rirvm/rir_mirror:master /opt/rir/build/release/bin/R
```

2 Contents of this Artifact

This section contains a detailed overview of all the parts contained in this artifact. Readers mainly interested in reproducing the results from our paper are advised to skip running commands in this section and instead focus on the instructions in the later "Step-by-Step" section. All containers combined use less than 8 Gb of disk space when extracted.

2.1 Containers

Next to the main container shown in the previous section, we also provide a benchmarking container based on the former. This one additionally includes our benchmark suite (for the published experiments at revision `dc8ee38`) and our fork of the ReBench benchmark runner. To locally run the benchmarks for Ř, you can therefore use the following command:

```
$CR run -it $RG/benchmark:$VS /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks \
  /opt/rir/build/release "e:PIR-LLVM -S"
```

Running the whole suite on a beefy machine takes about 30 minutes, CTL-C aborts. This command is expected to produce some warnings:

1. `GNU/FASTR VM dir does not exist` is a warning of our benchmark wrapper script `Setup/run.sh`, since this container only includes Ř.

2. To fix `Error: Process niceness can not be set` the above command has to be called with the `--privileged` flag.
3. `Warning: Low mean run time` indicates that a benchmark iteration takes less than 100ms. This happens in the microbenchmarks (which are not central to the claims in the paper) and also in benchmarks, where the first iterations (before warmup) are so slow, that we cannot increase the workload, without the benchmarking taking an unduly amount of time to run.

Please note that the immediate output from that command is the filtered and averaged output of the ReBench benchmark driver. These numbers are *not* the ones used in the paper. Instead we use the raw results exported to a file and analyze them independently. The exact command for running these experiments, the data export, analysis and visualization we refer to the later step-by-step section.

This container has a SHA2-256 of

`b164cb8f2f9da6428cf0695379a0361d87ebaaed4107436904da0dbfcb77f6f9`

A third container called benchmark-baseline contains our benchmarking infrastructure, the version of GNU R and the version of FastR used for the paper.

To run the suite with GNU R or FastR, use:

```
$CR run -it $RG/benchmark-baseline /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks . \
  /opt/rir/external/custom-r /opt/graal \
  "e:GNU-R -S"
$CR run -it $RG/benchmark-baseline /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks . \
  /opt/rir/external/custom-r /opt/graal \
  "e:FASTR -S"
```

Again, we refer to the step-by-step instructions on how to export and use the raw data.

This container has a SHA2-256 of

`70f9db5ed083ce9fb91661e3d6e3175cd88531aefb931ee857c10fd602d42a57`

2.2 Data Analysis

In the `data` subdirectory, you can find the following:

1. `data/plot.R` is the R script that produces the plots and outputs results.
2. `data/data` contains data files of the raw iteration times of the benchmark runs featured in the paper.

3. `data/results.tex` contains the results used in the paper, this corresponds to the output of running `plot.R` and includes every performance number in the paper text.
4. `data/final` contains the plots generated by `plot.R`.

The data files in `data/data` are the raw `data` files produced by ReBench in its ad-hoc format. In our evaluation, benchmark runs are triggered through a gitlab CI job and exported as build artifacts. The filenames correspond to job ids. For instance the reported performance numbers in section 5.2 were produced by job 545921038.

2.3 Source Code

The Ř source code, including source dependencies is included in the `src` directory. You can find instructions on how to build Ř at the end of this document.

The directory `benchmark-suite` contains all benchmarks.

2.4 Image Creation

Ř is continuously tested and benchmarked. Development of Ř happens in our main repository. For testing and benchmarking the repository is mirrored and for each commit a gitlab CI pipeline (see for example) is run. This pipeline does the following things:

1. Build a container with Ř and publish it tagged with the commit SHA.
2. Build a derived benchmark container which additionally includes benchmarking infrastructure.
3. Run all regression tests as well as the official GNU R test suite.
4. Run the benchmark suite on dedicated benchmark hardware and export the results as build artifact.

Normally old containers are cleaned up automatically. To prevent that we copied the containers for this artifact to a subhierarchy called `splash20` in the registry.

Instructions to reproduce the images follows in the next section.

3 Reproduction Step-by-Step

This section is a step-by-step recipe to reproduce all the data and recreate the analysis and results featured in the paper.

3.1 Prerequisites

First ensure that the correct ENV variables are set.

```
export CR=docker  # or podman, or 'sudo docker'
export VS=bc1933dde2673bf830f4505bb2483cd1fdd282ab
export RG=registry.gitlab.com/rirvm/rir_mirror/splash20
```

To recreate plots and data analysis it is useful to start with a fresh copy of this artifact and a copy of the original data for comparison.

```
tar xzf 502.tgz
cd oopsla20-artifact-502
cp -r data data-bkp
```

The following instructions assume that container images are received from our container registry. Alternatively they can be imported from the accompanying archive using `$CR load 502-images.tgz`.

3.2 Re-Create Containers

If you wish to re-build the containers yourself, then they are based on the Dockerfile at the root of our repository, and more Dockerfiles in the `container` directory. These steps are optional, you can use our already built containers. They take up to an hour, depending on the number of CPU cores available.

```
cd src
# build main container with GNU R and Ř
$CR build -t $RG:$VS .
# build container with benchmarking infra
cd container/benchmark
$CR build --build-arg CI_COMMIT_SHA=$VS -t $RG/benchmark:$VS .
# build container with GNU R and FastR
cd ../benchmark-baseline
$CR build -t $RG/benchmark-baseline .
cd ../..
```

3.3 Conformance

Our claim that Ř is passing GNU R regression tests is verified by:

```
$CR run -it $RG:$VS /opt/rir/build/release/bin/gnur-make-tests check-all
```

This command can take several hours to complete. The `gnur-make-tests` script is a simple wrapper around calling `make check-all` in the GNU R source directory, using Ř as the R command and also cleaning all left over files from previous test executions.

3.4 Rerun Performance Measurements

The following instructions are to run our benchmark suite in all the configurations featured in the paper. Note: to get consistent results these commands must be run in an idle machine, with background jobs (e.g., cron) disabled. These steps are optional, if you skip this subsection, then the later data analysis and visualization reuses our experimental data.

3.4.1 Data for Section 5.2

This section allows you to create your own performance measurements for the comparison between the Ř, GNU R and FastR VMs featured in section 5.2. Each of these runs takes about 30 minutes.

```
mkdir -p results

# Run test suite with Ř
$CR run --privileged \
  --mount type=bind,source=$PWD/results,destination=/media \
  -it $RG/benchmark:$VS /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks \
  /opt/rir/build/release "e:PIR-LLVM -S -df /media/benchmarks.data"

# Run test suite with GNU R
$CR run --privileged \
  --mount type=bind,source=$PWD/results,destination=/media \
  -it $RG/benchmark-baseline /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks . \
  /opt/rir/external/custom-r /opt/graal \
  "e:GNU-R -S -df /media/benchmarks-gnur.data"

# Run test suite with FastR
$CR run --privileged \
  --mount type=bind,source=$PWD/results,destination=/media \
  -it $RG/benchmark-baseline /opt/rbenchmarking/Setup/run.sh \
  /opt/rbenchmarking/rebench.conf /opt/rbenchmarking/Benchmarks . \
  /opt/rir/external/custom-r /opt/graal \
  "e:FASTR -S -df /media/benchmarks-fastr.data"

# Override our results with your reproduced data
cp results/benchmarks.data       data/data/545921038.data
cp results/benchmarks-gnur.data  data/data/545921041.data
cp results/benchmarks-fastr.data data/data/545921042.data
```

3.4.2 Data for Section 5.3

This section allows you to record performance data for the comparison of specialization levels featured in section 5.3 and export the results into a directory called `results`. Each of these runs takes about 30 minutes.

```
mkdir -p results

# Run the suite for each level
for i in {0..6}; do
  $CR run --privileged \
    --mount type=bind,source=$PWD/results,destination=/media \
    -e PIR_GLOBAL_SPECIALIZATION_LEVEL=$i -it $RG/benchmark:$VS \
    /opt/rbenchmarking/Setup/run.sh /opt/rbenchmarking/rebench.conf \
    /opt/rbenchmarking/Benchmarks /opt/rir/build/release \
    "e:PIR-LLVM -S -df /media/benchmarks-level-$i.data"
done

# Override our results with your reproduced data:
cp results/benchmarks-level-0.data data/data/545897134.data
cp results/benchmarks-level-1.data data/data/545898616.data
cp results/benchmarks-level-2.data data/data/545900400.data
cp results/benchmarks-level-3.data data/data/545901871.data
cp results/benchmarks-level-4.data data/data/545903708.data
cp results/benchmarks-level-5.data data/data/545905509.data
cp results/benchmarks-level-6.data data/data/546438259.data
```

3.5 Plots and Numerical results

This section allows you to recreate all graphs and recompute all performance numbers featured in the paper. If you recorded your own data in the previous section the results will of course differ from the results in the paper. To have meaningful numbers you need to run all the commands in one experiment in an idle machine. If you do not overwrite any performance measurements in `data/data`, then you are able to recreate the same graphs and results as featured in the paper. Due to a bug discovered after the submission the horizontal order of the individual measurements (small dots in the graphs) are different from the submission.

The following command plots all graphs into `data/final` and records all numerical results into `data/results.tex` using the performance data in `data/data`:

```
sudo apt-get install r-cran-ggplot2
# must be run in the data directory!
cd data
Rscript plot.R > results.tex
```

```
# compare results, numbers should be identical for unchanged data/data
# and in the same ballpark for clean reruns
diff results.tex ../data-bkp/results.tex
# visually compare plots in data/final...
```

4 Local Build

For completeness the `src` directory contains all the sources and external source dependencies to build Ř.

As prerequisites to build Ř we also need to build GNU R, which has a number of dependencies. Either install the required dependencies, by, e.g.:

```
apt-get install git libcurl4-openssl-dev texlive-latex-extra \
  texlive-latex-base texlive-fonts-recommended texlive-fonts-extra \
  texlive-latex-recommended texlive-font-utils dvipng cm-super bison \
  ca-certificates-java java-common libbison-dev libcairo-script-interpreter2 \
  libcairo2-dev libjbig-dev libmime-charset-perl libpango1.0-dev libpcsclite1 \
  libpixman-1-dev libsombok3 libtext-unidecode-perl libtiff5-dev libtiffxx5 \
  libunicode-linebreak-perl libxcb-render0-dev libxcb-shm0-dev \
  libxml-libxml-perl libxml-namespacesupport-perl libxml-sax-base-perl \
  libxml-sax-perl mpack openjdk-11-jre-headless texinfo g++ xdg-utils gfortran \
  subversion make r-base-dev liblzma-dev sed binutils curl cmake rsync \
  xorg-dev valgrind cppcheck xvfb xauth xfonts-base tk-dev ninja-build \
  python-pip flex bison make automake libgfortran3
```

Or perform the build in our base container `registry.gitlab.com/rirvm/rir_mirror/base` which is based on `ubuntu:18.04` and has all those packages pre-installed.

The build itself is performed as follows:

```
cd src
cmake -DCMAKE_BUILD_TYPE=release .
# Builds patched GNU R and downloads correct binary LLVM release
cmake --build . -- setup
# Update build scripts and builds Ř
cmake .
cmake --build .
# Run Ř
bin/R
```