Getting Started with Adapton's OOPSLA'15 Artifact

1. Install a Program to run the virtual machine
   - We used VirtualBox with default settings
   - Go to https://www.virtualbox.org/wiki/Downloads
   - Select the appropriate option for your platform
   - Install the program

2. Import Adapton's VM image file:
   - Load `paper193.ova` (http://www.cs.umd.edu/projects/PL/adapton-oopsla2015/paper193.ova).
   - From Menu: File->Import Appliance...

3. Run the VM and log on
   - Double-click the icon in the left pane
   - A window will open with Arch Linux (command-line only; there is no internal windowing system).
   - Hit enter or wait a few seconds to advance from the boot screen
   - Enter the user/password: guest/guest

     ```
     ic login: guest
     ```

     ```
     Password: guest (text hidden)
     ```

4. Run our test script:

   ```
   [guest@ic ~]$ ./recreate-results.sh
   ```

   - This will take 5-10 minutes

5. Inspect the generated tables:

   ```
   [guest@ic ~]$ less table1-results.txt
   ```

   ```
   [guest@ic ~]$ less table2-results.txt
   ```

The format of these two table files matches that of the tables in the paper (tables 1 and 2).

However, to generate these tables quickly, the `recreate-results.sh` script performs fewer samples and runs our benchmarks on smaller inputs than in the paper's tables. For this reason, the results from this script will not generally match the tables in the paper. In particular, our speedups generally **increase with increased input sizes**. (Instructions to create the full experiments are below).

We also welcome you to perform interactive tests with more demanding parameters, described further below.

All our results reported in the paper were from a 2.26 GHz Intel Mac Pro with 16 GB of RAM running Mac OS X 10.6.8. Our original data can be found in `oopals15_data_log/`. We measure the time taken to perform calculations both processor intensive and memory intensive. Results may vary greatly depending on the system used. The trends should not vary as much, so we encourage reviewers to consider speedups rather than time taken.

6. **(Optional)** Recreate the full experiments we reported in our paper.
   - Run the script above with the `full` option:

     ```
     recreate-results.sh full
     ```

   - Unlike the short version described above, this invocation of the script (using `full`) will take most of a day to run, and will require different VM settings. In particular, you'll need to change settings on the VM to allow 8 GB of memory (the default is 4 GB).
   - Our original raw data is in `oopals15_data_log/`
7. Let us know if you have any trouble!

Step by step guide to Adapton's OOPSLA'15 Artifact

The **Getting Started** guide above automatically collects experimental data and re-creates tables from the paper. This section explains how to run individual experiments manually, and explores the other content of the VM image.

Manual Experiments

List-based (and Tree-based) Tests

The test script we ran above is `script/test-oopsla15.sh`. This script begins with a list of options, which can be edited manually.

The following sequence will generate results:

```
[guest@ic ~]$ cd adapton.ocaml/
```

```
[guest@ic adapton.ocaml]$ ./experiment.native --experiment Rope_mergesort_lazyrecalc --0 --r --num-changes 1 --n 50000
```

```
[guest@ic adapton.ocaml]$ ./experiment.native --experiment Rope_mergesort_name --0 --r --num-changes 1 --n 50000
```

Try using `--help` for help:

```
[guest@ic adapton.ocaml]$ ./experiment.native --help
```

For more details, see our supplementary material.

The tests above consist of larger input sizes than those reported in the paper. Each invocation only performs a single sample. We see that Non-Adapton ('lazyrecalc') takes much less time to generate the initial list, but far more time to make an incremental change than Adapton ('name'). The output on screen is sufficient to see a difference; the full measurements can be found in an output file:

```
[guest@ic adapton.ocaml]$ cd out/
```

```
[guest@ic adapton.ocaml]$ tail experiments.csv
```

Here each test has a line for initial construction, one for each change made, and one for final heap size. The time is the second floating point number (the first one with some decimal digits).

IMP-based Tests

The imp test script that generates data for table2 is `test.py`. You can edit the common parameters at the top:

```
[guest@ic ~]$ cd ~/incremental-computation
```

```
[guest@ic incremental-computation]$ nano test.py
```

Nano has common commands listed at the bottom. Press control-x to exit and y [enter] to save your work.

The parameters at the top are set to run simpler tests that finish quickly.`array_size` and `intl_input` must be powers of 2. The second group of parameters are the ones used in the paper submission. This script is run as part of the test set above. You can call it as above, or add a parameter to run the longer set of tests `reproduce-results.sh full`. This will take many hours, though.

Tests can also be run individually. We don't suggest doing this unless you want to deal with the details of how our tests work, but the following sequence will produce results:

```
[guest@ic incremental-computations]$ ./imptests.native --artlib fromscratch --test fact3 --record --fact
10000
```

```
[guest@ic incremental-computations]$ ./imptests.native --artlib structural --test fact3 --record --fact 10000
```

```
[guest@ic incremental-computations]$ ./imptests.native --artlib nominal --test fact3 --record --fact 10000
```

```
[guest@ic incremental-computations]$ cd results
```

```
[guest@ic incremental-computations]$ tail fact-begin-swap-assign.csv
```

What you see is the raw output file. The last three lines are the ones we just generated. They are more demanding but less thorough then the paper's results. The first floating point number in each line is the initial program run time. The second floating point number is the incremental run time. As you can see, Adapton ('structural', 'nominal') takes slightly longer for the initial program than Non-Adapton ('fromscratch'). The benefit is in the incremental change runtime, where Adapton performs far better.

VM Contents

```
[guest@ic ~]$ ls
```

Our main Adapton git repo has been cloned to `adapton.ocaml/`.

```
[guest@ic ~]$ cd adapton.ocaml/
```

It is also available publicly: https://github.com/plum-umd/adapton.ocaml

Contents of `adapton.ocaml` directory:

- `script/` -- sample scripts for correctness and performance testing.
- `Source/` -- our implementation.

- `Sample/` -- example stub code for new projects.
- `out/` -- results from scripts, after they run.

The following is a clone of our development repository for the IMP interpreter:

```
[guest@ic adapton.ocaml]$ cd ~/incremental-computation
```

Contents of `incremental-computation` directory:

- `imptests.native` -- the testing executable (compiled OCaml code).
- `test.py` -- a helper testing script.
- `results/` -- results directory, with raw output measurements.
- `imp/` -- the OCaml code being tested.

Make targets

The `adapton.ocaml` and `incremental-computation` directories each have targets for `make`.

Targets for `adapton.ocaml`:

- `make test` -- rebuilds our test program `experiments.native`
- `make test-db` -- rebuilds our test program `experiments.byte` (bytecode with debugging symbols)
- `make opam-pin` -- builds adapton library and use `opam` to install it locally
- `make opam-reload` -- use after `opam-pin` has been run once

Targets for `incremental-computation`:

- `make` -- rebuilds our test programs `imp/imptests.native` and `imp/imptests.byte`