Verifying Spatial Properties of Array Computations

Artifact guide

Andrew Rice[1], Dominic Orchard[2], Matthew Danish[1], and Mistral Contrastin[1]

[1]Computer Laboratory, University of Cambridge
[2]School of Computing, University of Kent

August 29, 2017

This artifact contains the code necessary to reproduce our study of array programming idioms in scientific/numerical Fortran code (Section 2 of the paper), to demonstrate the inference, checking, and specification synthesis features of our verification tool via examples (Section 3 of the paper), and to produce statistics describing the various kinds of specifications that can be inferred over our test corpus (Section 7 of the paper).

In Section 2.1 of the paper we list the software packages that we included in our corpus. Due to licensing restrictions we can only include 6 of 11 in the artifact but in most cases the authors are willing to make them available through various mechanisms. We include a set of links and notes later for the source of each package.

Note, that all of our tools are open-source and can be found at:

- `https://github.com/camfort/camfort` - The main CamFort verification tool (also builds on our Fortran parsing/analysis library `https://github.com/camfort/fortran-src`)
- `https://github.com/camfort/array-analyse` - Static analysis tool used for Section 2
- `https://github.com/camfort/camfort-analyse` - Evaluation scripts used in Section 7

The revisions used in this artifact have been tagged with 'artifact2017' on their respective repositories.

Getting started

The artifact is provided as a VirtualBox appliance. We tested it using VirtualBox version 5.1.22 r11512 on macOS Sierra (10.12.15), El Capitan (10.11.6), and Linux (distribution Mint 17) but other modern versions of VirtualBox on other platforms should also work. A single user `camfort` with password `camfort` has been created.

1. **Install VirtualBox** by following the instructions at `https://www.virtualbox.org/wiki/Downloads` to install a version of VirtualBox for your operating system.

2. **Find and import the appliance** accompanying this guide using the option from the File menu in VirtualBox
3. **Start the VM** once it has finished importing
4. **Click the 'Log in' button** to sign in as camfort once the VM has booted
5. You should be presented with a terminal. Make sure you can run the `camfort` tool. For example, try

   ```
   camfort stencils-infer camfort/samples/stencils/simple.f90
   ```

   This should return the following output, informing the user of the inferred specifications for `simple.f90`:

   ```
   Inferring stencil specs for 'camfort/samples/stencils/simple.f90'

   camfort/samples/stencils/simple.f90
   (12:9)-(12:67)    stencil readOnce, (pointed(dim=1))*(centered(depth=1,
   ↪  dim=2)) + (centered(depth=1, dim=1))*(pointed(dim=2)) :: a
   ```

Step-by-step instructions

The CamFort repository contains sample files for each of the examples in the paper. Most of these are stored in `/home/camfort/camfort/samples/stencils`.

Laplace (Section 1 of the paper)

The example for the one-dimensional discrete Laplace transform was given in Section 1 of the paper.

1. Change to the samples directory: `cd /home/camfort/camfort/samples/stencils`
2. Inspect the source file: `less laplace.f90`
3. Use CamFort to infer the stencil specification: `camfort stencils-infer laplace.f90`

   ```
   Inferring stencil specs for 'laplace.f90'

   laplace.f90
   (14:2)-(14:32)    stencil readOnce, (centered(depth=1, dim=1)) :: a
   ```
4. From the output we see that on line 14, CamFort has determined that the array variable `a` is read with a centered spatial pattern of depth 1.
5. We can also use CamFort to update the source file itself with the new specification: `camfort stencils-synth laplace.f90 --inplace`
6. You can see the added specification (which appears within the `do` block, on line 14) by running `less laplace.f90` again.
7. Now that the array computation has a specification we can use CamFort to check the code against it: `camfort stencils-check laplace.f90`

8. We can now try introducing an indexing error into the source code and see if it is detected by CamFort as conflicting with the stencil specification. Open the file (e.g. use `gedit laplace.f90`) and change the array index `i-1` on line 15 to `i+1`.

9. Check the code against the specification using `camfort stencils-check laplace.f90`

```
Checking stencil specs for 'laplace.f90'

laplace.f90
(14:2)-(14:53)    Not well specified.

       Specification is:

              stencil readOnce, (centered(depth=1, dim=1)) :: a

       but at (15:2)-(15:32) the code behaves as

              stencil (forward(depth=1, dim=1)) :: a
```

10. From the output we can see that the code behaviour has been inferred as "forward" rather than "centered" and so a verification error is returned.

Navier (Section 1 of the paper)

Section 1 of the paper also refers to code from a Navier-Stokes fluid model. A Fortran implementation of this model has been checked out into `/home/camfort/corpus/navier/fortran`.

The example in Section 1 of the paper gives the computation to compute `du2dx` from `u`, `duvdy` from `u` and `v` and so on. The implementation of this computation is given from lines 22 to 35 in `/home/camfort/corpus/navier/fortran/simulation_mod.f90`. After inspecting the file you can use CamFort to synthesise the stencil specifications:

```
cd /home/camfort/corpus/navier/fortran
camfort stencils-synth --inplace .
```

(Note that the trailing 'dot' is significant: this indicates to CamFort that it should process all the files in the current directory.)

The contents of `simulation_mod.f90` should now have been updated to contain various stencil specifications (including two on lines 35 and 36) — view this with `less simulation_mod.f90` or `gedit simulation_mod.f90`. These specifications are attached to the computation of `f(i,j)` (which is the array assignment) and refer to the variables `u` and `v` which flow into the assignment through `du2dx`, `duvdy` etc.

Small synthesis examples

Sections 2 and 3 of the paper contain a variety of small examples demonstrating the features of our specification language. These are stored in `/home/camfort/camfort/samples/stencils` and can be used with CamFort, e.g.,

```
camfort stencils-infer *filename*
```

Or to synthesis an inferred specification inplace

```
camfort stencils-synth --inplace *filename*
```

The examples in order are:

1. `decomposed-three-part.f90` (Section 2.1 of the paper) shows how CamFort will track data flow in order to derive the correct specification for an array update. Try `camfort stencils-infer decomposed-three-part.f90` to see the inferred specification.
2. `single-region.f90` (Section 3 of the paper) demonstrates five single region stencils (i.e. specifications do not use the region combination operators `+` and `*`). Note that the inferred specifications here get an extra qualifier `readOnly` which was not explained at the point in the paper where this example first appears (for the purpose of introducing the language incrementally). Try `camfort stencils-infer` or `camfort stencils-synth --inplace` on this file.
3. `combined-region.f90` (Section 3 of the paper) shows a nine-point stencil specification using the intersection operator `*` and a five-point stencil specification built using the union operator `+`.
4. `region-declaration.f90` (Section 3 of the paper) shows a declaration for the Roberts Cross stencil pattern which can be reused within a program for clarity. Try using `camfort stencils-check` to validate the code against the specification already given in this file.
5. `approx.f90` (Section 3 of the paper) cannot be specified exactly with the specification language and so `camfort stencils-infer` will infer a pair of `atLeast` and an `atMost` specifications to bound its behaviour.
6. `relativisation.f90` (Section 3 of the paper) contains an array update to a subscript with a non-zero offset (i.e. the update is at `i+1` rather than `i`). CamFort is able to generate a stencil specification that is 'relativised' to this left-hand side offset (try `camfort stencils-infer samples/stencils/relativisation.f90`).
7. `access.f90` (Section 3 of the paper) does not contain a stencil computation. Instead, it contains an array computation of the maximum element in an array. CamFort therefore will infer/synth an `access` specification for this, distinguishing the programming pattern from a stencil computation.

Empirical study of array computations (Section 2 of the paper)

The design of our specification language is based on an empirical study of array computations in Fortran code (see Section 2 of the paper). This is based on a separate static analysis tool that we built using CamFort as a library. A software corpus of eleven numerical or scientific packages were analysed as part of this study.

The software corpus

Where possible we have included packages from our corpus in `/home/camfort/corpus`. Below is the complete list of packages that we used, their source, and a description of any modifications we made (if any).

Included

- GEOS-Chem (http://acmg.seas.harvard.edu/geos/), tropospheric chemistry model. The source code for this model is available at https://bitbucket.org/gcst/geos-chem.

  Commit SHA: `3071d9b722d8d07a97c4a10189044ddb57d8f7ec`.

- Navier, a small-size Navier-Stokes fluid model. Source code is available at https://github.com/dorchard/navier.

  Commit SHA: `10f6f6e7b56a3d53c3f49145a6bc20e457ef6fc9`.

- BLAS, a common linear-algebra library used in modelling available from http://www.netlib.org/blas/. We used version 3.6.0.

- ARPACK-NG, an open-source library for solving large-scale eigenvalue problems available from https://github.com/opencollab/arpack-ng.

  Commit SHA: `0e2ed1b2f2f3636a8dbdba580006c9dbe536e717`.

  We patched for standards compliance with `/home/camfort/patches/arpackng.patch`.

- SPECFEM3D, global seismic wave models, source code is available at https://github.com/geodynamics/specfem3d.

  Commit SHA: `48c539b730dcdaa845bad930d5790e5cdbbba5c7`.

  We patched for standards compliance with `/home/camfort/patches/specfm3d.patch`.

- Cliff's, a tsunami model, source code is available at https://github.com/Delta-function/cliffs-src.

  Commit SHA: `137583958c37ce466b81ce2ec1611fbd64c16ac5`.

  We patched for standards compliance with `/home/camfort/patches/cliffssrc.sh`.

Not included but can be downloaded

- MUDPACK, a general multi-grid solver for elliptical partial differentials. Download from https://www2.cisl.ucar.edu/resources/legacy/mudpack/download

- Computational Physics (CP), programs from a popular textbook "An Introduction to Computational Physics". The sample programs from the book are available here: http://www.physics.unlv.edu/~pang/cp_f90.html

Not included, requires special license agreement

- The Unified Model (UM) (http://www.metoffice.gov.uk/research/modelling-systems/unified-model) developed by the UK Met Office for weather modelling. Source code is available through https://code.metoffice.gov.uk. Requires a strong license agreement. There is a contact address on that page with details of how to apply for access.

- E3ME (https://www.camecon.com/how/e3me-model/), a mixed economic/energy impact prediction model. This is a commercial model developed by Cambridge Econometrics (https://www.camecon.com/). Send inquiries with a project proposal to the company to request a license for the source code.

- Hybrid4, a global scale ecosystem model. The development of this model is led by Dr Andrew Friend (http://www.geog.cam.ac.uk/people/friend/) at the University of Cambridge. Inquiries (with a project proposal) should be directed to him.

We are in active collaboration with the developers of these packages.

Generating study data and analysing a single package

1. Change to the corpus directory: `cd /home/camfort/corpus`
2. Perform an analysis on one of the corpus packages, e.g., navier, by running:

   ```
   array-analysis navier
   ```

   (you may notice warnings `NON-HANDLED-RANGE`, these are just debug messages from inside our analysis tool).
3. This generate a summary of the results and a data file `navier.restart`. Summary data can be reviewed at any time using `array-analysis VIEW rfile` where there is a file `rfile.restart`. In this case, try running:

   ```
   array-analysis VIEW navier
   ```
4. This raw data file can be analysed/categorised to generate data matching the format of the different data analyses in Section 2.2 of our paper by using the `study` program:

   ```
   study navier.restart
   ```

   This generates tables of data matching the categorisations used in Section 2.2.

   Note that study takes file name arguments with `.restart` endings; whilst `array-analysis` should not have `.restart` in the file names. Apologies for this quirk. If you do `array-analysis VIEW navier.restart`, then you will get a blank data set as the tool will try to read `navier.restart.restart` which does not exist.

Generating study data and analysing the provided corpus

Note that the results of the paper are based on the aggregated data of all 11 packages, so the exact number wont be reproducible unless all packages are available. However, the 6 provided packages (and easily downloaded additional 2 packages) can be analysed, providing data which exactly supports the conclusions of Section 2 of the paper.

1. Change to the directory: `/home/camfort/corpus`
2. Perform an analysis and study on the whole corpus directory by running

   ```
   array-analysis-per-directory.sh
   study combined.restart
   ```

This could take time, e.g. about 2 hours on a desktop/laptop with 2.7GHz Core i5, 8GB RAM.

(Note: there are some files with non-standard code in the corpus which are not parsed/lexed by our tool. Don't be alarmed by parsing/lexeing errors appearing during such a run. The analysis tool will skip these.)

Alternatively you can perform the analyses of each corpus package in parallel with

```
array-analysis-per-directory.sh par
```

Evaluating CamFort (Section 7 of the paper)

In Section 7 of the paper we summarize the various kinds of specifications found in our corpus by running the tool in inference mode. These are generated by the programs in the `camfort-analyse` repository.

1. Change to the directory: `cd /home/camfort/camfort-analyse`
2. Start a run (expected run-time about 1-2 hours) by typing: `specification-analysis.sh` (or by `./eval/infer.sh`)

   Regarding memory usage: if you are not able to allocate enough memory, you can direct `specification-analysis.sh` to run on one package from the corpus at a time by setting the CORPUS environment variable as in one of the following examples:

   ```
   export CORPUS=/home/camfort/corpus/arpack-ng
   export CORPUS=/home/camfort/corpus/BLAS-3.6.0
   export CORPUS=/home/camfort/corpus/cliffs-src
   export CORPUS=/home/camfort/corpus/geos-chem
   export CORPUS=/home/camfort/corpus/navier
   export CORPUS=/home/camfort/corpus/specfem3d
   ```
3. Examine the output of the most recent run: `./eval/last.sh`
4. The log files are all stored in the `./eval/logs/` directory with a timestamped filename so that you can examine them directly.
5. Obtain a summary of a log file by running: `camfort-analyse < log-file` (where `log-file` is the path to one of log files in `/home/camfort/camfort-analyse/eval/logs/`).
6. The summary will contain key-value pairs for each corpus repository as well as an aggregation of the data headed `overall` at the end.

Most key-value pairs are simply a count of the number of times that keyword appears in the log-file. Some of the key-value pairs have a number at the end, in which case they are a count of how many times that keyword appears paired with that number in a specification. For example, the keyword `depth1` counts how many specifications have `depth=1` in the log-file.

The full list of relevant keys is as follows:

dimTagN is the number of arrays with dimensionality N.

dimsN is the number of specifications with N dim= terms in them.

emptySpec is the number of potential array computations which had no specification.

inconsistentIV is the number of potential array computations which used induction variables inconsistently.

justPointed is the number of specifications which were pointed only.

lexFailed is the number of files that failed in the lexer.

lexOrParseFailed is the number of files that failed in the lexer or parser.

linesParsed is the number of lines of code successfully parsed.

linesTotal is the number of lines of code processed whether successfully or not.

plusOpsN is the number of specifications with N union + operators.

mulOpsN is the number of specifications with N intersection * operators.

regionOpsN is the number of specifications with N region operators (of either kind).

multiAction is the number of specifications with more than one action.

multiActionRegionOpsN is the number of multiAction specifications with N region operations (of either kind).

nonNeighbour is the number of specifications inferred for array computations involving non neighbour indices.

numStencilLines is the number of lines generated with specifications.

numStencilSpecs is the number of array variables given stencil specifications.

parseFailed is the number of files that failed to parse.

parseOk is the number of files that successfully parsed.

sadepthN is the number of specifications of depth N with only a single action.

singleAction is the number of specifications with only a single action.

singleActionIrr is the number of specifications with only a single non-pointed action.

somePointed is the number of specifications with some dimensions pointed.

tickAssign is the number of potential sites for stencil specifications.

tickAssignSuccess is the number of potential sites which successfully had a specification inferred.

LHSnotHandled is the number of specifications for which CamFort cannot handle the LHS.

relativized is the number of specifications where offsets had to be shifted in order to infer them correctly (relativisation)

In the paper we report the following quantities all of which are defined by a particular key in the summary:

- **All stencil (and access) specifications:** numStencilSpecs
- **Specifications which are pointed only:** justPointed
- **Single-action specifications** (one forward, backward or centered region combined with any number of pointed regions)**:** singleAction
- **Single-action with a non-pointed modifier:** singleActionIrr
- **Multi-action specifications** (at least 2 forward, backward or centered regions combined with any number of pointed regions — these represent interesting, more complex spatial patterns in code)**:** multiAction
- **Multi-action only using intersection (*):** multiActionMulOnly
- **Specifications with N intersection (*) operators:** mulOps0, mulOps1, mulOps2, etc.

- **Specifications with N union (+) operators:** plusOps0, plusOps1, plusOps2, etc.
- **Upper-bounded specifications:** atMost
- **Lower-bounded specifications:** atLeast
- **readOnce specifications:** readOnce

The overall results for the analysis can be seen at the end of the data generated by `camfort-analyse < log-file` under the heading overall.

The exact numbers for Section 7 paper rely on the whole corpus of 11 packages. However, by running on the subset of the corpus that we are able to provide openly, the same conclusions can be drawn: that indeed, the specifications provided by the CamFort are highly applicable to numerical computing programs, and that the choice of combinators in the specification language is well supported by the data.