APmap User Manual

Jintao Yu

27th August 2020

1 Introduction

APmap is a compiler that compiles a user's applications into the configurations for automata accelerators such as Cache Automaton[1] and RRAM-AP.[2] The compiler targets a two-level switching network that consists of global and local switches.

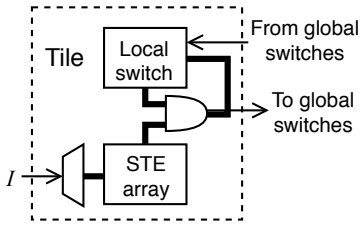

Figure 1: A tile

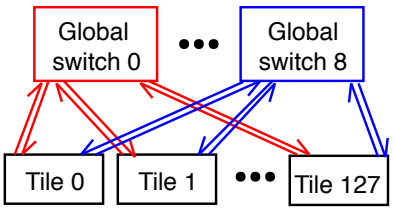

Figure 2: Routing network

The default parameter of APmap models the switching network of Cache Automaton. It consists of 128 tiles that each contain a local switch and a state-transit element (STE) array, as shown in Figure 1. Cache Automaton also has eight global switches that each link with all the tiles, as shown in Figure 2.

The network parameters can be configured in APmap via *def.h*. The configurable values include the number of tiles, the number of global switches, the number of STEs in a tile. With this feature, you can generate the configuration fit for your need or explore the hardware design space.

2 Build from the source

2.1 Dependency

APmap calls METIS [3] to partition the target automata. Therefore, you need to install METIS before building APmap.

[1] Arun Subramaniyan et al. "Cache Automaton". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-50 '17. Cambridge, Massachusetts: ACM, Oct. 2017, pp. 259–272. URL: http://doi.acm.org/10.1145/3123939.3123986.

[2] J. Yu et al. "Memristive devices for computation-in-memory". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2018, pp. 1646–1651. DOI: 10.23919/DATE.2018.8342278.

[3] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

2.2 Build on Linux

A simple “make" command will build APmap. It has been tested on Ubuntu, CentOS, and Cygwin.

2.3 Build on other OS

It should not be challenging to build this project on any mainstream operating system since the source code is written in C.

3 Getting started

3.1 Input format

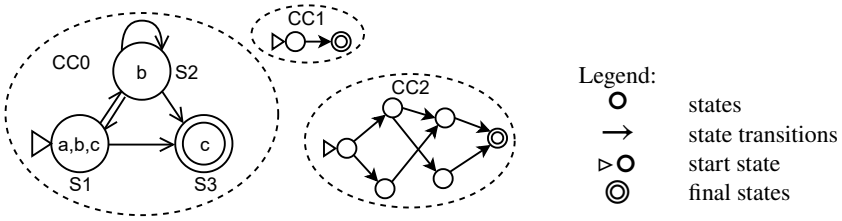

Figure 3: An automaton that consists of three connected components.

The top-level input file is with the suffix *.map*. An automaton can be divided into one or more non-overlapping parts without transitions between any two parts. These parts are named as *connected components* (CCs). For example, Figure 3 contains three CCs, i.e., $CC0$, $CC1$, and $CC2$. It can be described by the following file:

```
1 % automata.map
2 3
3 3 5 cc0.graph
4 2 1 cc1.graph
5 6 7 cc2.graph
```

Any line that starts with a % symbol is comment and will be omitted by APmap. The first line that is not comment specifies the number of CCs in the automaton. From the next line, every line lists three key features of a CC, i.e., the state number, state transition number, and the name of the *.graph* file that describes the details.

Every line of the *.graph* file describes a state, including the state name, whether it is a start or final state, STE configuration, and the state transitions. The *cc0.graph* mentioned in the last *.map* file represents $CC0$ in Figure 3.

```
1 % cc0.graph
2 S1 1 0 00000000 00000000 00000000 E0000000 00000000 00000000 00000000
       00000000 2 3
3 S2 0 0 00000000 00000000 00000000 40000000 00000000 00000000 00000000
       00000000 1 2 3
4 S3 0 1 00000000 00000000 00000000 20000000 00000000 00000000 00000000
       00000000
```

$S1$ presented in line 2 is a start state (the '1' after 'S1'), but not a final state (the '0' after '1'). The STE configuration field consists of eight eight-digit hexadecimal numbers that are generated based on the input symbols that the state accepts.[4] The last part of a line consists of several line indexes (comment lines are not counted) that the state transits to. In the case of $S1$, it transits to the states described in line index 2 (i.e., $S2$) and line index 3 (i.e., $S3$). If a state has no outgoing transitions such as $S3$, the state transition field is empty.

These files are placed in the *test* folder. You can run this test using the command:

```
$ cd test
$ ../apmap automata.map
```

3.2 Output format

After the execution, a file named *map_result* will be generated in the folder. It describes the hardware configuration of the automata processor. For the example presented in the last section, only a tile is used. The last lines are:

```
281 ...
282 252: S4 0 0 00000000 00000000 00000000 80000000 00000000 00000000
        00000000 00000000 -> 250
283 253: S3 0 0 00000000 00000000 00000000 90000000 00000000 00000000
        00000000 00000000 -> 252
284 254: S2 0 0 00000000 00000000 00000000 A0000000 00000000 00000000
        00000000 00000000 -> 252 251
285 255: S1 1 0 00000000 00000000 00000000 B0000000 00000000 00000000
        00000000 00000000 -> 254 253
```

Each line describes an STE. The first field is the index, which is between 0 and 255. The following fields are almost the same as those in the *.graph* files, except that the destination STEs are referred to using STE indexes.

The configuration of global switches is similar. Nevertheless, it is not shown in this example, as no global switches are used.

3.3 Toolchain integration

For large automata, it would be difficult for composing the input files manually. Therefore, we recommend a stack flow with other tools. RAPID [5] is a high-level programming language developed for automata processing. It can be used to describe the application, and its compiler generates ANML files as the output. VASim [6] can parse ANML files and conduct optimizations. We modified VASim to provide the input files for APmap. The toolchain is shown in Figure 4.

The modified VASim files are placed in the *vasim* folder, i.e., *main.cpp*. We add an option "--apmap" (short form: "-A") for VASim to generate the *.map* and *.graph* files for APmap.

[4] P. Dlugosch et al. "An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing". In: *IEEE Transactions on Parallel and Distributed Systems* 25.12 (2014), pp. 3088–3098. ISSN: 1045-9219. DOI: 10.1109/TPDS.2014.8.

[5] https://github.com/kevinaangstadt/rapid

[6] https://github.com/jackwadden/VASim

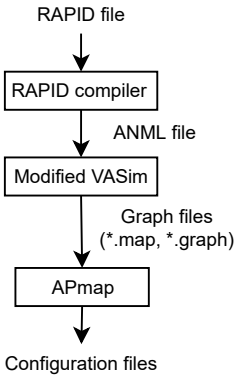

Figure 4: Recommended toolchain.

This modification is based on the latest version of VASim (version 4c615cb). You can download the source code of VASim, replace *main.cpp* with the one we provide and compile. To convert automata to the APmap format, you can use the following command:

```
$ vasim –A xxx.anml
```

Other options, such as “-O", can also be applied together with “-A".