Technical Disclosure Commons

Defensive Publications Series

October 2020

Programmable Hardware Accelerator For Regular Expression Queries

N/A

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
N/A, "Programmable Hardware Accelerator For Regular Expression Queries", Technical Disclosure Commons, (October 29, 2020)
https://www.tdcommons.org/dpubs_series/3726

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Programmable Hardware Accelerator For Regular Expression Queries

ABSTRACT

Regular expression (regex) queries are used extensively in data analytics applications. Hardware-based regex searches can make searches efficient across a variety of application domains. However, hardware accelerators that can support arbitrary regular expressions are currently infeasible due to the very large number of possible states and state transitions. This disclosure describes techniques to map an input regular expression to a non-deterministic finite automaton and hardware such as FPGA or ASIC that can be programmed to filter an input data stream to search for arbitrary (customer-given) regular expressions.

KEYWORDS

- Regular expression
- Regex query
- Regex matching
- Hardware accelerator
- Deterministic finite automaton (DFA)
- Non-deterministic finite automaton (NFA)

BACKGROUND

Regular expression (regex) queries are used extensively in data analytics applications, e.g., to search through data, to match patterns, to match logs, to crawl websites, in search indexing, etc. Regular expression query filtering is usually performed by mapping the regular expression query to either a deterministic finite automaton (DFA) or non-deterministic finite automaton (NFA). Implementing DFA and NFAs in hardware such as a field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC) for a single regular expression is

relatively straightforward. However, DFA/NFA hardware accelerators that can support arbitrary regular expressions are currently considered infeasible due to the very large number of possible states and state transitions.

DESCRIPTION

This disclosure describes techniques to map an input regular expression to an NFA and a hardware design such as a field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC) that can be programmed to filter an input data stream to search for an arbitrary (customer-given) regular expression.

Per the techniques, a regular expression, e.g., one that is to be matched on a large data set, is converted into an NFA using standard techniques. The states and state-transitions of the NFA are converted into programmable values in a regex accelerator. The regex accelerator has programmable logic that has as activations the NFA that is programmed into it.

At this initial state after programming, the NFA has no states that are active; its output is therefore zero. The input data that is to be filtered is broken into tuples and streamed through the filter. Tuples are individual units of data, each comprising one or more ASCII characters. The inflow of ASCII characters into the regex accelerator causes state transitions to occur. The output result logic monitors for the required state to be activated in order to provide a positive result. The output is ready after all the characters of the tuple have passed through the filter.

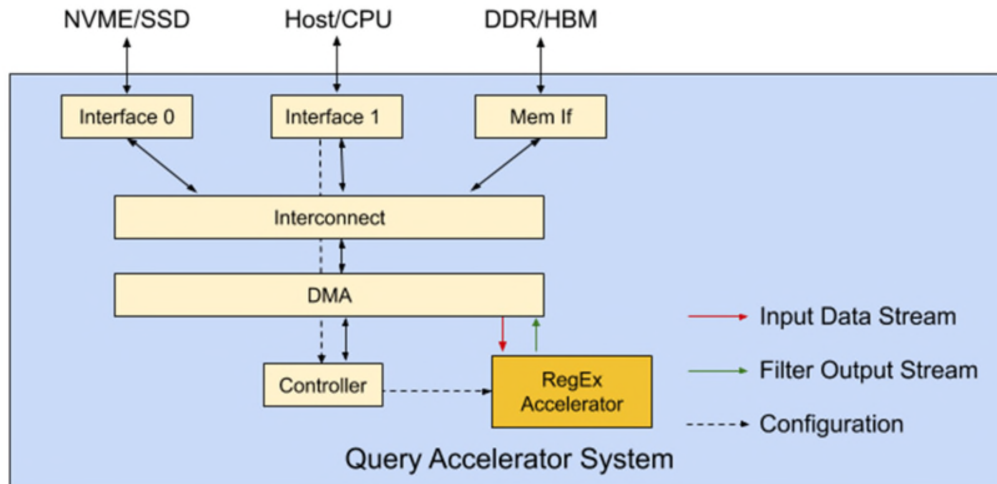

Fig. 1: Programmable hardware accelerator for regular expression queries

Fig. 1 illustrates an example schematic for a programmable hardware accelerator for regular expressions, in particular, the query accelerator, per the techniques of this disclosure. The query accelerator includes the following components.

- High speed interfaces such as peripheral component interconnect express (PCIe) to off-chip or off-FPGA components such as memories, CPU, etc. that are communicate with the query accelerator
- On-chip (or on-FPGA) interconnect used to move data through different components of the accelerator.
- Regular expression hardware accelerator (regex accelerator), which receives input data from the direct memory access (DMA) and filters the input as per the required regular expression. The regex accelerator is described in greater detail below.
- DMA engine for data moves to and from the regex accelerator.
- Controller used to configure and control the regex accelerator.

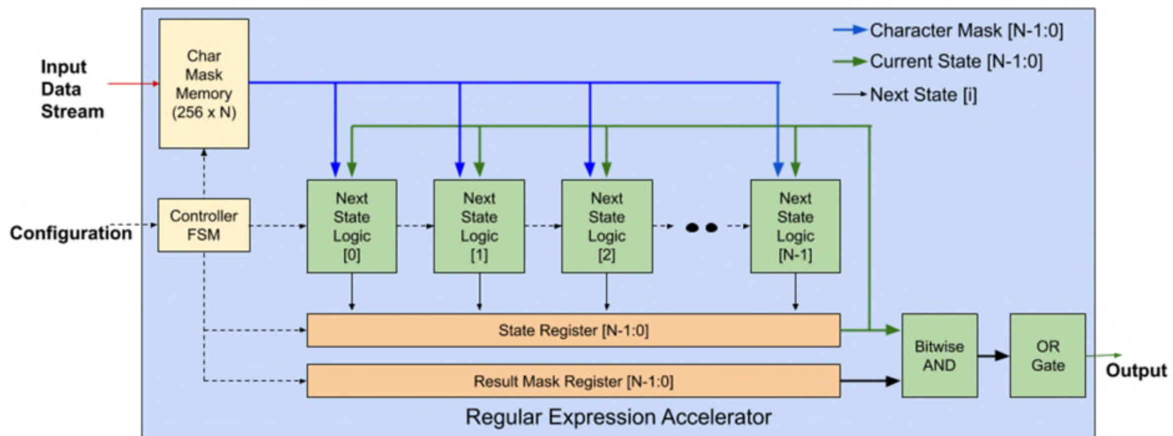

Fig. 2: Regex accelerator

Fig. 2 illustrates an example schematic for the regex accelerator. The regex accelerator includes the following components:

- **Controller FSM:** The controller FSM is used to forward the configuration data to appropriate locations. It is also used to control the next state logic and allow for epsilon propagation.
- **Next state logic blocks:** These are logic blocks that compute the next state of the NFA when a character of a tuple is received. The number of these blocks is equal to the number of states, *N*, of the biggest NFA that the accelerator can handle. For example, if the accelerator is to be capable of handling an NFA with 500 states, then the number of next state logic blocks is 500. When processing an NFA with less than 500 states, only a subset of the next state logic blocks are active. The next state logic block is described in greater detail below.
- **State register:** This is a register to hold the current state of the NFA. The total bit-width of this register equals *N*. After a character is received, the output of every next state logic is flopped into the corresponding bit of the state register.

- **Result mask register:** This is a programmable register that indicates the state of the NFA that provides a positive result. It is applied to the state register (using a bitwise AND gate) to obtain the result. For example, consider an NFA with five states such that the result is positive if the fifth state is activated. In this case, the fifth bit of the result mask register is 1, e.g., when state 5 activates, the output of the regex accelerator is 1 for this tuple.
- **Character mask memory:** This is a 256-entry memory (ASCII characters are 8 bits, e.g., 2^8=256 in number). Each entry corresponds to a valid ASCII character. The bit-width of an entry is *N*. A bit *i* of an entry "*m*" is 1 if the NFA's state[*i*] is 1 when the character "*m*" is received. In other words, if character "*m*" is received, the states that are activated have 1 in the corresponding bits of the "*m*"th entry of the character mask memory. This memory is read when a character is received, and the read data is provided to the next state logic blocks for computation of the next state. For example, suppose state 3 is 1 if character "A" is received. The ASCII value of "A" is 65. Therefore, in the 65th entry of the memory, the 3rd bit is 1.

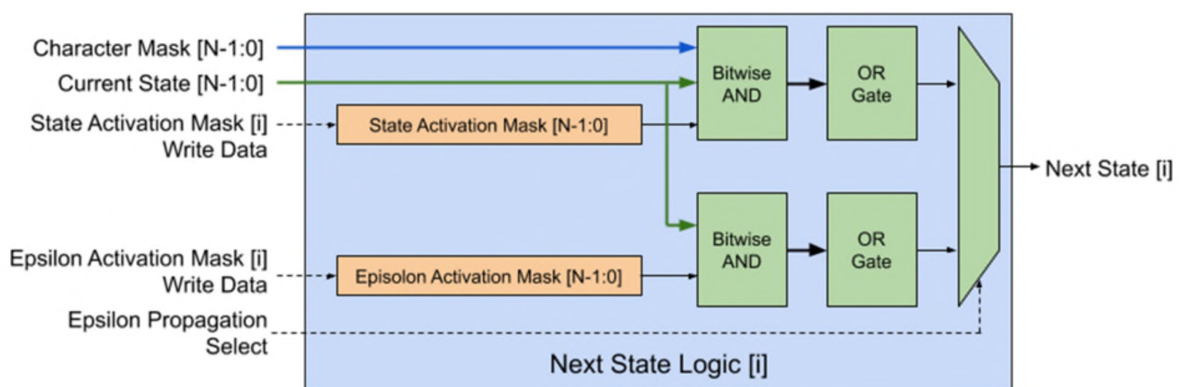

Fig. 3: Next state logic

Fig. 3 illustrates an example schematic for the next state logic block. The next state logic block includes the following components:

- **Input interfaces** for the current state [$N-1$:0] (from the state register) and the character mask [$N-1$:0] (from the character mask memory).
- **State activation mask register:** This register represents all the states that have arcs feeding into the given state. In other words, this register represents the states that, if active, cause the given state to also be active when the appropriate character is received. This is applied to the current state along with the character mask in order to determine if there is an activation to 1. For example, suppose there is an arc from state 3 to state 5 and this arc activates when character "C" is received. In the state activation mask for the next state logic of state 5, bit 3 is 1. Suppose the current state of state 3 is a 1. Because state 3 is a 1, and because there is an arc from state 3 to state 5 (as programmed in the state activation mask), the next state of state 5 will also be a 1 if character "C" is received. The character mask determines if a character "C" was received by having bit 5 set. Therefore, when the character mask and the state activation mask are applied to the current state, the next value of this state is known.
- **Epsilon transition mask register:** Epsilon transitions are special transitions that can be thought of as arcs that have no character needed for activation. For a given state, the epsilon transition mask register can be used to compute the next state value based on epsilon transitions. For example, if there is an epsilon transition from state 1 to state 3, then if the current value of state 1 changes to 1, the current value of state 3 should also change to a 1. For this example, in the next state logic of state 3, bit 1 of the epsilon transition mask register is set to 1.

- **Multiplexer:** A multiplexer is provided that selects between the epsilon-transitioned next state value and the character activated next state value. This is because every next state logic computation is split into two phases for every character in the following order - the controller finite state machine (FSM) controls the phases and aligns them with the reading of the character mask from the memory:
    - **Character-based state activation:** When a character is received, based on the current state, the character mask, and the state activation mask, the next state is computed. This value is updated into the state register.
    - **Epsilon-based state activation:** After character-based state activation, the updated value of the state register is subjected to epsilon transitions if any. This value is also updated into the same state register. The next state logic is then ready to receive the next character.

Example use cases

The disclosed regex hardware accelerator can efficiently run regex queries on large data sets with millions or billions of records, with speeds approximately 100-200 times faster than traditional general-purpose multicore CPUs. The power consumed (and the cost) of running such regex queries on very large datasets is a fraction of the power consumed by traditional CPUs. As long as the datasets comprise alphanumeric data, the data can be from a variety of application domains, e.g., census data, protein patterns, logs, data from crawled websites, search indexing, etc.

The following is an example application of the disclosed regex hardware accelerator applied to census data:

1. **Setup:** The disclosed regex hardware accelerator is implemented, e.g., in an FPGA or an ASIC, is attached to a server with a general-purpose CPU, as shown in Fig. 1. The accelerator is attached to the CPU using a high-speed connection such as PCIe.
2. **Query:** An example query is as follows: *match all records that start with "100," that have an apartment number, and that are in cities of Austin or Round Rock in the state of Texas*. The query accelerator is configured to run the regex matching on the "address" field of each record.
3. **Configuration:** The CPU converts the above query to an NFA and then into the actual values that are programmed into the registers (result mask register, character mask registers, state activation mask registers, epsilon transition mask registers, etc.). The CPU also programs the addresses of the location of the to-be-filtered data set and the address where the output has to be written. This is programmed into the controller for the accelerator. The CPU notifies the accelerator to start processing.
4. **Hardware operation:** The controller creates DMA requests to read the data into the accelerator. As the data starts to stream into the accelerator, the regex accelerator processes the characters of the tuples (in this case, the "address" field of each record is a tuple) and indicates a match/no-match after each tuple. This result is recorded as an output. The number of results in the output matches the number of input tuples. The output of the accelerator is streamed back to a destination address specified by the CPU.
5. **Performance:** The regex accelerator can intake one character per 2 clock cycles. Assuming that the "address" field of the record (the tuple) is about 1000 characters and that the accelerator is running at 1.5GHz, the accelerator can run regex queries on approximately 0.75 million records per second. Assuming that there are 330 million

records in the US Census data set (matching the population of the United States), the accelerator can run a regular expression query on the entire data set in approximately seven minutes. The same task is likely to take multiple days or weeks to run on a state-of-the-art multi-core general-purpose CPU. Note that if the tuple size, which is application dependent, is smaller than 1000, it can process more records per second thus yielding higher performance.

CONCLUSION

This disclosure describes techniques to map an input regular expression to a non-deterministic finite automaton and hardware such as FPGA or ASIC that can be programmed to filter an input data stream to search for arbitrary (customer-given) regular expressions.

REFERENCES

[1] Gonzalo Navarro and Mathieu Raffinot, “Fast and Simple Character Classes and Bounded Gaps Pattern Matching, with Applications to Protein Searching”, *Journal of Computational Biology*, Vol. 10, No. 6, 903-923.

[2] R. Sidhu and V. K. Prasanna, "Fast Regular Expression Matching Using FPGAs," *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, Rohnert Park, CA, USA, 2001, pp. 227-238.