Scalable FSM Parallelization via Path Fusion and Higher-Order Speculation

Junqiao Qiu\(^1\), Xiaofan Sun\(^2\), Amir Hossein Nobleh Sabet\(^2\), and Zhijia Zhao\(^2\)

\(^1\)Michigan Technological University, \(^2\)University of California, Riverside

1. Motivation

As a basic computation model, finite-state machine (FSM) embodies a variety of important applications, ranging from intrusion detection [21, 11, 18, 3] and data decoding [10, 17] to motif searching [16, 4], rule mining [19], and textual data analytics [13, 6, 5]. Unfortunately, the execution of an FSM is known to be “embarrassingly sequential” [2, 23], due to the inherent dependences among state transitions – in each state transition, the current state always depends on the prior state \(^1\). These state dependences fundamentally limit the performance of FSM-based computations on modern processors, where parallelism plays an increasingly critical role.

2. State of The Art

To address the inherent dependences in FSM computations, prior work [7, 9, 23, 12, 22, 14, 15] has mainly followed two basic parallelization schemes: state enumeration and state speculation (see Figure 1-b). Assume the input to an FSM (e.g., a binary sequence) is partitioned evenly into two chunks, as shown in Figure 1-a. Due to the dependences among state transitions, the starting state for the second chunk would be unknown, until the first chunk has been processed – the ending state of the first chunk is the starting state of the second chunk. To process the two chunks in parallel, one can choose:

(i) **State Enumeration.** As the unknown starting state must be one of the states in the FSM, we can enumerate all of them by forking an execution path for each state [7, 12]. Obviously, maintaining all the execution paths may bring significant overhead. To reduce it, prior work [12] checks if some paths transition to the same state, known as path merging, in which case only one of them needs to be kept. However, the effectiveness of this approach highly depends on the state convergence property of the FSM. When some of the execution paths exhibit slow convergence or fail to converge, the overhead of this scheme would be high.

(ii) **State Speculation.** Instead of considering all the states, one can guess the starting state of the second chunk [23, 22, 14, 15]. To ensure correctness, the predicted state must be validated against the ending state of the prior chunk – the ground truth. If the validation fails (i.e., misspeculation), the chunk needs to be reprocessed. However, when the input is partitioned into multiple chunks, the ending state of the prior chunk may not be the ground truth until its own speculation has been validated (with needed reprocessing).

\(^{1}\)This work was performed when the author was a Ph.D. student at UCR.

\(^{2}\)Here, an FSM refers to a deterministic finite automaton (DFA).

---

**Figure 1:** FSM Parallelization: Challenges and Solutions

These serialized validations form a fundamental scalability bottleneck in the speculative FSM parallelization [15].

In addition, a hybrid scheme may enumerate a subset of states [8, 20], which makes a tradeoff between the limitations of both schemes. In summary, the existing FSM parallelization schemes face fundamental scalability challenges.

3. Major Contributions

This work introduces two novel techniques: path fusion and higher-order speculation, to address the scalability challenges in the two basic FSM parallelization schemes, respectively.

3.1. Path Fusion

For state enumeration, we propose to fuse different execution paths into a single path to lower down the overhead.

**Intuition.** An interesting observation we made is that state enumeration suffers from a similar kind of inefficiency as the execution of nondeterministic finite automaton (NFA). The former needs to maintain a vector of states for all the execution paths, while the latter needs to track a subset of active states. A well-known solution to the inefficiency of NFA execution is to convert the NFA to an equivalent DFA (deterministic finite automaton) using the subset construction algorithm [1]. Can we design a similar technique to address the inefficiency in state enumeration? In fact, we find that, by developing a vector construction algorithm similar to the subset construction algorithm, we can generate a new FSM whose single execution path mimics multiple execution paths of the original FSM. We call this technique path fusion.

**Static Path Fusion.** The key to path fusion is to construct a fused FSM where each state corresponds to a vector of states...
in the original FSM. Like NFA to DFA conversion [1], we can statically construct the fused FSM. First, we map the initial fused state to state vector \([S_0, S_1, \ldots, S_N]\) which corresponds to the enumerated execution paths. Then, we feed every input symbol to the existing fused states to iteratively discover new fused states and valid fused state transitions, until no new fused states can be found. In theory, the fused FSM can be very large as it traverses a space of \(N^N\), where \(N\) is the size of the original FSM. However, in practice, their sizes are often well below \(N^3\) and even \(N^2\). Despite the promises, it might still be desired that the fused FSM can fit into a memory budget.

**Dynamic Path Fusion.** Unlike static path fusion which builds the entire fused FSM for all possible inputs, dynamic path fusion constructs a partial fused FSM that only consists of states and transitions for a single input. The idea of dynamic path fusion resembles the just-in-time (JIT) compilation used in modern compilers. It consists of two execution modes: the basic mode where different paths are enumerated and fused state transitions are generated, and the fused mode which only makes fused state transitions. An execution starts from the basic mode, then switches between the two modes based on the availability of the fused state transitions.

### 3.2. Higher-Order Speculation

To address the serial validation bottleneck in state speculation, we introduce the concept of speculation order.

**Speculation Order.** Formally, we denote the speculation at the beginning of input chunk \(i\) as:

\[
\text{SPEC}(i, S, C)
\]

where \(S\) is the predicted starting state and \(C\) is the correct starting state, also referred to as the correctness criterion. By feeding a speculated correctness criterion to the speculation, we can raise the speculation to higher orders. For example,

\[
\text{SPEC}^{k+1}(i, S, C) \xrightarrow{\text{validate}} \text{SPEC}^k(i, C, C')
\]

means that a \(k + 1\)-th order speculation, after its validation, becomes a \(k\)-th order speculation. The correctness criterion of the former becomes the predicted state of the latter.

Based on the above formalization, it is not hard to find that all prior FSM speculation techniques [9, 23, 22, 14, 15], in fact, belong to first-order speculation, as the correctness criteria used in their validations are always non-speculative.

**Benefits of Higher-Order Speculation.** Raising the order of speculation may bring benefits in two aspects:

- Chunks with higher-order speculation no longer need to wait for the ground truth, thus can be validated earlier;
- The validation of higher-order speculation introduces a new speculated state which, in theory, is more likely to the correct starting state, thus improving the accuracy.

<table>
<thead>
<tr>
<th>Table 1: Speedup Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Baseline: sequential execution; #threads: 64; input size: (4 \times 10^8))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FSM</th>
<th>Seq(s)</th>
<th>B-Enum</th>
<th>B-Spec</th>
<th>S-Fusion</th>
<th>D-Fusion</th>
<th>H-Spec</th>
<th>BoostFSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>7.45</td>
<td>13.7</td>
<td>1.9</td>
<td>30.9</td>
<td>25.1</td>
<td>17.8</td>
<td>30.9</td>
</tr>
<tr>
<td>M2</td>
<td>7.48</td>
<td>29.1</td>
<td>20.7</td>
<td>29.8</td>
<td>19.6</td>
<td>32.6</td>
<td>32.6</td>
</tr>
<tr>
<td>M3</td>
<td>7.39</td>
<td>14.2</td>
<td>1.4</td>
<td>30.8</td>
<td>25.1</td>
<td>18.3</td>
<td>30.8</td>
</tr>
<tr>
<td>M4</td>
<td>7.43</td>
<td>11.1</td>
<td>0.6</td>
<td>31.1</td>
<td>25.5</td>
<td>13.9</td>
<td>31.1</td>
</tr>
<tr>
<td>M5</td>
<td>7.43</td>
<td>28.5</td>
<td>22.9</td>
<td>-</td>
<td>13.1</td>
<td>30.1</td>
<td>30.1</td>
</tr>
<tr>
<td>M6</td>
<td>7.57</td>
<td>26.9</td>
<td>21.6</td>
<td>-</td>
<td>16.1</td>
<td>32.6</td>
<td>32.6</td>
</tr>
<tr>
<td>M7</td>
<td>7.49</td>
<td>29.8</td>
<td>29.7</td>
<td>-</td>
<td>25.5</td>
<td>32.7</td>
<td>32.7</td>
</tr>
<tr>
<td>M8</td>
<td>7.46</td>
<td>13.0</td>
<td>39.8</td>
<td>30.9</td>
<td>24.9</td>
<td>39.2</td>
<td>39.8</td>
</tr>
<tr>
<td>M9</td>
<td>7.44</td>
<td>11.6</td>
<td>0.6</td>
<td>-</td>
<td>23.9</td>
<td>10.4</td>
<td>23.9</td>
</tr>
<tr>
<td>M10</td>
<td>7.37</td>
<td>7.3</td>
<td>1.9</td>
<td>-</td>
<td>8.5</td>
<td>13.0</td>
<td>7.5</td>
</tr>
<tr>
<td>M11</td>
<td>7.47</td>
<td>12.9</td>
<td>0.6</td>
<td>31.2</td>
<td>23.6</td>
<td>17.6</td>
<td>31.2</td>
</tr>
<tr>
<td>M12</td>
<td>7.53</td>
<td>12.9</td>
<td>0.5</td>
<td>-</td>
<td>3.6</td>
<td>8.7</td>
<td>12.9</td>
</tr>
<tr>
<td>M13</td>
<td>7.40</td>
<td>12.2</td>
<td>0.6</td>
<td>-</td>
<td>22.5</td>
<td>16.7</td>
<td>22.5</td>
</tr>
<tr>
<td>M14</td>
<td>7.46</td>
<td>12.7</td>
<td>0.9</td>
<td>-</td>
<td>23.5</td>
<td>11.2</td>
<td>23.5</td>
</tr>
<tr>
<td>M15</td>
<td>7.35</td>
<td>13.0</td>
<td>0.6</td>
<td>-</td>
<td>23.4</td>
<td>17.1</td>
<td>23.4</td>
</tr>
<tr>
<td>M16</td>
<td>7.51</td>
<td>19.3</td>
<td>37.2</td>
<td>-</td>
<td>17.9</td>
<td>36.5</td>
<td>37.2</td>
</tr>
<tr>
<td>Geo</td>
<td>-</td>
<td>15.4</td>
<td>3.1</td>
<td>31.0</td>
<td>18.3</td>
<td>19.5</td>
<td>25.8</td>
</tr>
</tbody>
</table>

the FSM computations into a series of iterations, gradually improving the speculation in a naturally parallel manner.

### 3.3. Scheme Selection

Together, we consider five FSM parallelization schemes:

- **B-Enum**: basic state enumeration
- **B-Spec**: basic state speculation
- **S-Fusion**: state enumeration with static path fusion
- **D-Fusion**: state enumeration with dynamic path fusion
- **H-Spec**: higher-order (iterative) speculation

Which scheme works the best depends on the characteristics of the FSM and its inputs. We design a set of heuristics, as a decision tree, to guide the scheme selection.

### 4. Key Results

We integrated the above FSM parallelization schemes along with the scheme selector into a multi-scheme parallelization framework, named BoostFSM. Table 1 reports the speedups of different schemes over the sequential FSM execution on a 64-core machine. The FSMs are collected from a widely used open-source network intrusion detection system (Snort), carrying diverse properties. First, S-Fusion is found feasible for five FSMs, for which it raises the speedups from \(12.9 \times\) (B-Enum) to \(31.0 \times\) on average. Next, D-Fusion shows varying speedups, from \(3.6 \times\) to \(25.5 \times\), as its efficiency highly depends on the skewness of the fused state transitions and the state vector size. Third, comparing to B-Spec, H-Spec shows consistent improvements (from \(3.1 \times\) to \(19.5 \times\)), thanks to its two benefits mentioned earlier. Finally, the last column reports the results of scheme selection. Among 16 FSMs, it successfully finds the best scheme for 15 FSMs. The failed case is due to our coarse-grained performance modeling.

### 5. Why ASPLOS

This work fits ASPLOS for its focus on the parallelization and scalability of a fundamental class of computations and for its uses of speculation and compiler techniques.
References


