Enabling Scalability-Sensitive Speculative Parallelization for FSM Computations

Junqiao Qiu, Zhijia Zhao
UC Riverside
jqiu004@ucr.edu, zhijia@cs.ucr.edu

Bo Wu
Colorado School of Mines
bwu@mines.edu

Abhinav Vishnu,
Shuaiwen Leon Song
Pacific Northwest National Lab
Abhinav.Vishnu@pnnl.gov,
Shuaiwen.Song@pnnl.gov

ABSTRACT

Finite state machines (FSMs) are the backbone of many applications, but are difficult to parallelize due to their inherent dependencies. Speculative FSM parallelization has shown promise on multicore machines with up to eight cores. However, as hardware parallelism grows (e.g., Xeon Phi has up to 288 logical cores), a fundamental question raises: *How does the speculative FSM parallelization scale as the number of cores increases?* Without answering this question, existing methods for speculative FSM parallelization simply choose to use all available cores, which might not only waste computing resources, but also result in suboptimal performance.

In this work, we conduct a systematic scalability analysis for speculative FSM parallelization. Unlike many other parallelizations which can be modeled by the classic Amdahl's law or its simple extensions, speculative FSM parallelization scales unconventionally due to the non-deterministic nature of speculation and the cost variations of misspeculation. To address these challenges, this work introduces a spectrum of scalability models that are customized to the properties of specific FSMs and the underlying architecture. The models, for the first time, precisely capture the scalability of speculative parallelization for different FSM computations, and clearly show the existence of a "sweet spot" in terms of the number of cores employed by the speculative FSM parallelization to achieve the optimal performance. To make the scalability models practical, we develop S3, a scalability-sensitive speculation framework for FSM parallelization. For any given FSM, S3 can automatically characterize its properties and analyze its scalability, hence guide speculative parallelization towards the optimal performance and more efficient use of computing resources. Evaluations on different FSMs and architectures confirm the accuracy of the proposed models and show that S3 achieves significant speedup (up to 5X) and energy savings (up to 77%).

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; • **Computer systems organization** → **Parallel architectures**;

KEYWORDS

FSM, Scalability, Speculation, Parallelization, Finite State Machine

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICS '17, June 14-16, 2017, Chicago, IL, USA

© 2017 Association for Computing Machinery.

ACM ISBN 978-1-4503-5020-4/17/06...\$15.00

<https://doi.org/http://dx.doi.org/10.1145/3079079.3079082>

ACM Reference format:

Junqiao Qiu, Zhijia Zhao, Bo Wu, and Abhinav Vishnu, Shuaiwen Leon Song. 2017. Enabling Scalability-Sensitive Speculative Parallelization for FSM Computations. In *Proceedings of ICS '17, Chicago, IL, USA, June 14-16, 2017*, 10 pages.
<https://doi.org/http://dx.doi.org/10.1145/3079079.3079082>

1 INTRODUCTION

Scalability is fundamental to the high-performance applications. An accurate scalability analysis helps realize the optimal performance and avoid unnecessary use of additional computing resources. This work is to provide an accurate scalability analysis for the speculative parallelization of finite state machine (FSM) computations.

As a classic computation model, FSMs have been widely used in many critical applications, such as intrusion detection [16, 36], data decoding [14, 34], genome and protein motif searching [4, 33], and high-performance web analytics [24]. For its fundamental role in many performance-critical applications, it is anticipated that emerging architectures will feature hardware supports for FSM computations, such as automata processor [6].

However, due to the tight dependencies among state transitions, FSM computations are extremely difficult to parallelize. As shown in the code snippet below, at each transition, the current state `state` not only depends on the input symbol `c` but also the prior state `prior`. Such dependencies essentially form a dependence chain, inherently prevent any FSM computations from running in parallel.

```
prior = init; c='';
while c!=EOF do {
    c = read();
    state = trans(prior, c); // dependence
     $\phi$ (state, c); // action at a state
    prior = state; }
```

State of the Art. To overcome the dependencies, existing methods often rely on speculative parallelization [22, 26, 38]. Basically, they first partition the input sequence evenly into N_{core} chunks where N_{core} is the number of available cores, then process the chunks in parallel, each with a predicted starting state, except the first chunk. In the case where a prediction fails, they need to reprocess the wrong part to ensure the correctness (see Section 2). This strategy has shown promise on small-scale multicore processors (up to eight cores). However, it is still poorly understood how well it can scale to larger parallel platforms with tens of or even hundreds of processing cores¹, such as Xeon Phi processors.

In particular, *is it always the best practice to use all available cores (i.e., N_{core}) to achieve the best performance? If not, what is the optimal number of cores to employ to maximize the benefits of speculative FSM parallelization?*

¹By default, this work refers to logical cores as cores, unless noted otherwise.

Figure 1: Suboptimal Speedup of Existing Methods [38] on Larger-scale Platforms (256-core Xeon Phi).

Without answering these questions, existing methods may not only suffer from suboptimal performance, but also waste computing resources that would be otherwise used for other computations. As illustrated by Figure 1, when executed on a Xeon Phi processor with 256 logical cores, existing methods [37, 38] result in suboptimal speedup on two FSM benchmarks, up to nearly 5X performance degradation comparing to the optimal ones. It is also important to note that the optimal number of cores varies across different FSMs.

Unlike prior work that focus on either designing speculation techniques [38] or reducing profiling cost [37], this work aims to achieve the optimal performance gain for speculative FSM parallelization with an accurate scalability analysis.

However, accurately analyzing the scalability for speculative FSM parallelization is challenging for four-fold reasons. First, by nature, speculative parallelization is non-deterministic. Its overall performance highly depends on the accuracy of the speculation. Second, when a speculation fails (*misspeculation*), it is required to reprocess the incorrect part. But the processing cost may vary across different chunks, depending on the convergence. Consequently, the total cost of misspeculation correlates with the number of cores non-linearly, making existing models fail to capture its scalability. Finally, the actual scalabilities of FSM computations are also constrained by the machines where they are executed via resource contention and relative execution speed.

To address the above complexities, this work introduces a series of scalability models for speculative FSM parallelization. The models integrate a probabilistic analysis to capture the non-deterministic behaviors of speculation and an offline *sample-based conditional regression* (SCR) technique to characterize the cost variation of misspeculation. Unlike existing FSM characterization [37, 38] that requires to profile the convergence property for every pair of states, SCR only profiles state pairs that are more likely to appear in the actual speculative execution. Based on the probabilistic models and SCR, this work designs both architecture-independent scalability analysis and architecture-aware scalability analysis. The former analyzes the scalability solely based on the design of speculative parallelization and the properties of an FSM. It guides the designers to tune speculative parallelization scheme and helps developers compare the scalabilities of various FSMs. In comparison, the latter further characterizes the architecture factors that may affect the actual scalability, making the scalability analysis practical in real-world computing environments.

To effectively leverage the above scalability analyses, this work develops S3 – a scalability-sensitive speculative parallelization framework for FSM computations. At high level, S3 works in three steps: (1) it first characterizes the FSM’s properties and measures the architectural factors of the machine; (2) With the measurements, S3 next automatically reasons about the scalability and infers the optimal number of cores n^* to use; (3) Finally, it feeds n^* into the speculative parallelization to maximize its performance gain.

Experiments on a set of real-world FSM benchmarks demonstrate the accuracies of the proposed models and show that S3 can boost the performance of existing techniques up to 5X, with significant energy savings in most cases (up to 77%).

In sum, this work makes the following four-fold contributions.

- This work, for the first time, points out the suboptimality of existing speculative FSM parallelization methods when moving to larger-scale parallel platforms.
- It provides a series of rigorous scalability models, with a *sample-based conditional regression* technique. Together, they enable an accurate characterization of complex scaling behaviors in speculative FSM parallelization.
- To facilitate the use of the proposed models, it designs S3, a scalability-sensitive speculative parallelization framework that automatically reason about the scalability and guides the configuration of speculative parallelization at runtime.
- It evaluates S3 on both many-core processor (Xeon Phi processor) and multi-socket multi-core architectures, and demonstrates large ratio of performance improvements and significant energy reduction.

2 MOTIVATION

In this section, we first illustrate the basic approach of speculative parallelization used by existing work for FSM computations, then point out the suboptimality of performance in existing solutions due to their unawareness of scalability, hence the necessity to enable scalability-sensitive speculative parallelization.

Speculative Parallelization. To address the tight dependence among state transitions in FSM computations, existing solutions often rely on speculative parallelization techniques, which is based on a predict-validate-reprocess strategy. Next, we describe the high-level ideas of speculative parallelization of FSM computations, which consists of three major phases.

- (1) **Partition:** Divide the input sequence into equal-sized chunks according to the total number of CPU cores N_{core} .
- (2) **Predict & Process:** For each chunk i , predict its starting state s_{pred}^i and assign a thread to process ².
- (3) **Validate & Reprocess:** Once every thread has finished its chunk, check if the predicted state equals to the true state (i.e., $s_{true}^i = s_{pred}^i$) one by one. If a prediction fails, reprocess its corresponding chunk before validating the next starting state. Note that the reprocessing may stop earlier when states s_{true}^i and s_{pred}^i both transition to the same state (known as *state convergence*, see Section 4.1).

According to the first phase, the basic speculative parallelization approach assumes that it can scale up to the total number of CPU cores N_{core} . While this might be true for small-scale multicore processors (e.g., quad/oct-core processors), but may not hold for larger-scale platforms with tens of or hundreds of CPU cores.

Performance Suboptimality. Figure 2 shows the speedup curve for an FSM benchmark on a Xeon Phi machine with 256 cores. As the blue line shows, the actual speedup increases linearly at the beginning before reaching about 10 cores, which is confirmed by prior work [37, 38]. However, the increase becomes non-linear thereafter and even starts dropping after about 30 cores. Finally, the speedup drops to merely 5.7X when all 256 cores are used. This

²More details about the design of the prediction can be found in prior work [37, 38].

Figure 2: Projected Scalability v.s. Actual Scalability

result clearly demonstrates that using all available cores may not lead to the optimal performance.

On the other hand, unlike many parallel applications, the speedup curve of speculative FSM parallelization is difficult to model using the classic Amdahl's law and its simple extensions. As Figure 2 shows, the speedup curve predicted by Amdahl's law, with 3% of serial execution (green line), follows an obviously different trend comparing with actual speedup curve. The principle reason to such a discrepancy is due to the inherent complexities of speculative FSM parallelization. First, during a speculative FSM execution, not only the parallel part (i.e., Phase 2) depends on the number of cores, but also the sequential execution part (i.e., Phase 3). In comparison, the serial part in Amdahl's law, by default, is assumed to be a constant. Furthermore, the relation between the sequential execution performance and the number of cores in parallel part is non-linear, due to the variation of state convergence. The two complexities make existing scalability models fail to faithfully capture the scalability of this advanced parallelization technique.

To address the challenges and seek for the optimal performance, we propose S3, a speculative parallelization framework that can automatically characterize the scalability of a given FSM and calculates the best configuration to maximize the performance. We next give an overview of S3 before presenting its details.

3 OVERVIEW

At high level, S3 includes three layers. From bottom to top, they are *characterization*, *modeling* and *guidance*, as shown in Figure 3. We next briefly present each of the three layers in order.

Characterization. As FSMs and the underlying architecture affect the scalability of speculative parallelization from totally different ways, it is natural to separate their characterizations. We refer to the characterization results on the application side and architecture side, symbolically, as *FSM.properties* and *Arch.properties*, respectively.

① On one side, FSMs exhibit dramatically different behaviors when executed speculatively. Some FSMs are easier to speculate while others may be much more challenging (such as *div* in [38]), depending on their transition structures and input characteristics. Furthermore, when a misspeculation happens, the penalty not only varies across FSMs, but also varies across speculatively processed chunks, depending on how fast the predicted (wrong) state s_{pred} converges with the correct starting state s_{true} . The faster they converge, the less penalty the misspeculation incurs.

Prior work [37, 38] introduce metrics like *state feasibility* and *expected convergence length* to tune the design of the predictor. They are inadequate to accurately model the details of the non-deterministic behaviors of speculation, such as the distribution of misspeculation penalty across different chunks.

In comparison, S3 maintains a short list of raw convergence length samples (typically < 150) for each state pair, rather than simply averaging them. The list of samples encapsulates not only

Figure 3: Overview of S3

the average convergence length, but also its distribution, which is the key to accurately model the variation of misspeculation penalty (Section 4.2). To reduce the overhead of characterization, unlike existing methods which require to profile the convergence property for every pair of states, S3 only profiles the state pairs that are more likely to appear in actual speculative executions. We will elaborate FSM characterization and its uses in Section 4.

② On the other side, the characteristics of the architecture also directly affect the scalability of speculative FSM parallelization in various ways, depending on the specific design of the architecture. In this work, we focus on two main factors that play critical roles in the scalability analysis: *resource contention* and *relative execution speed*. Resource contention happens when different threads share the same computing resources, such as last level cache (LLC) and memory bandwidth. Depending on the design of the architecture and the number of concurrent threads, such contention could vary significantly. Note that resource contention only happens in the parallel phase of speculative FSM execution. When moving into the reprocessing phase, only a single thread is left due to dependencies, the contention hence reduces to zero. However, due to the tracking of state convergence, the execution speed in the reprocessing phase might be slightly slower than the parallel phase. This difference directly influences the scalability, but may vary across architectures. Therefore, it is necessary to capture the relative execution speed between the two phases, in order to precisely quantify the scalability. We will present architecture characterizations in Section 5.1.

Scalability Modeling. In this work, the scalability is defined as the capability of speculative parallelization to scale up to larger amount of computing units (i.e., CPU cores). In particular, given an FSM with a fixed-size input, the scalability concerns how the execution time varies with the number of CPU cores used ³.

③ With the characterization results, S3 can automatically reason about the scalability using a series of scalability models that are derived based on the design of speculative FSM parallelization.

In specific, the speedup S is defined as a non-linear function of the number of cores employed n , along with other parameters, such as the properties of FSM computations *FSM.properties* and the architecture properties *Arch.properties* ⁴. Depending on if *Arch.properties* is considered, the models fall into two types:

architecture-independent scalability model:

$$S = f(n, \text{FSM.properties}) \quad (1)$$

architecture-aware scalability model:

$$S = f'(n, \text{FSM.properties}, \text{Arch.properties}) \quad (2)$$

³This is commonly referred to as *strong scaling*.

⁴Note that some architecture properties, such as resource contention, also depend on the number of cores used n .

where n is the number of cores used in speculative execution, $FSM.properties$ represents convergence properties of the given FSM, and $Arch.properties$ contains architecture characteristics such as resource contention among threads and relative execution speed of different FSM operations. The architecture-independent models can be used to compare the scalabilities of different FSMs and guide the design of speculative parallelization; The architecture-aware models provide more accurate scalability analysis results that are customized for a specific architecture.

At high level, the entire speculative FSM execution time T_{spec} is broken down into two parts: the *parallel processing time* T_{para} and the *sequential reprocessing time* T_{repr} . Let T_{seq} be the sequential execution time, then the speedup of speculative parallelization can be defined as follows:

$$S = \frac{T_{seq}}{T_{spec}} = \frac{T_{seq}}{T_{para} + T_{repr}} \quad (3)$$

Unlike the classic Amdahl's law that assumes a constant ratio for the sequential part, in Equation 3, both the parallel part T_{para} and the sequential part T_{seq} primarily depend on the number of cores n . Moreover, the relation between T_{repr} and n follows a non-linear pattern, making standard scalability models fail to faithfully capture its scalability. We address the challenges with a novel *sample-based conditional regression* (SCR) technique. Different from traditional regression models, SCR conditionally accept convergence length samples based on the parameters of speculative parallelization. With such fine-grained customization, SCR can precisely model the above non-linear relation.

We will describe the basic scalability analysis in Section 4.2 and the integration of architecture properties in Section 5.2.

Scalability-Sensitive Speculative Parallelization. The goal of S3 is to maximize the efficiency of speculative parallelization by reasoning about its scalability and discovering the optimal number of cores to use (i.e., n^*).

④ With the scalability models, this problem can be formalized as the following discrete optimization problem.

$$\begin{aligned} \max S \\ \text{s.t. } 1 \leq n \leq N_{core} \end{aligned} \quad (4)$$

where the number of cores used by speculative FSM parallelization is bounded by the total number of cores on the machine. When $n = 1$, the FSM execution becomes sequential.

To solve the optimization problem, depending on the models, S3 either simply enumerates each configuration and chooses the one with the highest speedup, or directly computes the optimal configuration from a closed-form expression. By setting n^* in the speculative parallelization, S3 maximizes the speedup. We refer to this scheme as *scalability-sensitive speculative parallelization*.

In sum, the three layers closely depend on each other from top to bottom. They together enable a new speculative parallelization scheme for FSM computations on larger-scale parallel platforms. In the following, we elaborate architecture-independent scalability analysis and architecture-aware scalability analysis, respectively.

4 ARCHITECTURE-INDEPENDENT SCALABILITY ANALYSIS

This section presents the scalability analysis that does not assume any particular architecture, but solely based on the properties of the FSM computations and the design of speculative parallelization.

Figure 4: Example of State Convergence in FSM Transitions

4.1 FSM Characterization

As a basic computation model, FSMs feature many properties. In this work, we focus on a type of characteristics that has a significant influence on the penalty of misspeculation – the *convergence length*.

As mentioned in Section 2, when a misspeculation happens, the speculative parallelization framework may not have to reprocess the whole chunk, thanks to the fact that the predicted (wrong) state s_{pred} may converge with the actual state s_{true} . The shorter it takes for them to converge, the less penalty of the misspeculation incurs. To effectively model such behaviors, we leverage the concept of state convergence length [38], defined as follows.

Definition 4.1. Given an input string I and two different starting states s_i and s_j . The convergence length between s_i and s_j on I is the least number of transitions for each of them to take in order to transition to the same state, denoted as $L^I(s_i, s_j)$. If by end of I , they end at different states, set $L^I(s_i, s_j) = \infty$.

In Figure 4, though starting from two different states, transition paths s_1 and s_4 reach the same state s_2 after consuming the third input symbol, hence the convergence length between s_1 and s_4 on this piece of input is $L^I(s_1, s_4) = 3$.

Given an FSM, its convergence length properties can be profiled either offline using a set of training inputs [38], or online using the testing inputs [37]. Note that prior work require to profile the average state convergence length for every pair of states, in order to guide the design of the starting state predictor [37, 38]. In comparison, S3 maintains a pool of raw convergence length samples for state pairs that are more likely to appear in actual speculative executions, in order to facilitate a high-precision scalability analysis, as explained in the next subsection.

4.2 Scalability Analysis

The goal of scalability analysis is to examine how speedup S varies as the number of cores used n changes. Based on the definition of S in Equation 3, this requires to model the ratio between sequential execution time and its corresponding speculative execution time. In architecture-independent scalability analysis, we use the number of state transitions to quantify the relative execution time, instead of the concrete execution time which may vary across different architectures. For example, given an input of length $\|I\|$, when processed sequentially, the execution time is $T_{seq} = \|I\|$.

For speculative execution, the total time T_{spec} mainly consists of the parallel processing time T_{para} and the sequential reprocessing time T_{repr} (Phases 2 and 3 in Section 2)⁵.

$$T_{spec} = T_{para} + T_{repr} \quad (5)$$

In parallel processing phase, each thread first predicts the starting state, then processes its corresponding chunk with the predicted starting state. In general, there is a tradeoff between the prediction accuracy and prediction cost. However, after the design of predictor is fixed, the prediction cost becomes a constant (more details in

⁵The partitioning time in Phase 1 is typically negligible.

[37, 38]). We use C_{pred} to represent the prediction cost and T_{proc} to represent the processing time of chunks. Since the input is evenly partitioned based on the number of cores n , we have $T_{proc} = \|I\| / n$. Hence, the parallel phase execution time T_{para} is

$$T_{para} = C_{pred} + T_{proc} = C_{pred} + \frac{\|I\|}{n} \quad (6)$$

Next, we analyze the execution time of the reprocessing phase, which is more challenging due to two complexities inherited in the design of speculative FSM parallelization.

Complexity I: Undeterministic behaviors of speculation. By its nature, speculation is non-deterministic. If a speculation succeeds, there would be no cost of reprocessing; otherwise, the speculation framework has to initiate reprocessing to correct the mistakenly processed parts. As only the latter case degrades the scalability, an effective scalability model needs to distinguish the two cases. However, since the speculation happens during the actual runs, such a distinction is as hard as the speculation itself.

Complexity II: Variation of reprocessing costs. To reduce the penalty of misspeculation, existing methods [37, 38] leverage the convergence property of FSMs (see Section 4.1) by tracking if the misspeculated state s_{pred} converges with the actual state s_{true} . Once they converge, the reprocessing can safely stop. On one hand, this design helps reduce the reprocessing costs of misspeculation. On the other hand, it also complicates the modeling of speculative parallelization, as the reprocessing costs for different chunks may vary significantly, depending on their convergence lengths.

In the following, we present an analytical model that address both complexities together, referred to as *sample-based conditional regression model*. Before introducing the model, we first formalize the total execution time of reprocessing.

In reprocessing phase, due to dependencies, all chunks, except the first one, have to be validated and reprocessed sequentially. Therefore, the total reprocessing time T_{repr} is composed of the reprocessing time of each chunk T_{repr}^i , where $1 < i \leq n$. Let $L^i(s_{pred}^i, s_{true}^i)$ be the convergence length between the misspeculated state s_{pred}^i and the actual state s_{true}^i for chunk i . Then the total reprocessing time can be represented as

$$T_{repr} = \sum_{i=2}^n T_{repr}^i = \sum_{i=2}^n L^i(s_{pred}^i, s_{true}^i) \quad (7)$$

Two points worth to mention here. First, to address the above two complexities together, we unify the representations by referring to a successful speculation as a “misspeculation” with reprocessing length of zero, that is,

$$L^i(s_{pred}^i, s_{true}^i) = 0, \text{ if } s_{pred}^i = s_{true}^i \quad (8)$$

Second, the reprocessing of a chunk cannot go beyond the size of the chunk, hence the following constraint holds:

$$L^i(s_{pred}^i, s_{true}^i) \leq \frac{\|I\|}{n} \quad (9)$$

Sample-based Conditional Regression. A key challenge in the scalability analysis of speculative FSM parallelization is precisely estimating Equation 7 in practice. We address this challenge with *sample-based conditional regression* (SCR). Different from a classic regression analysis, SCR considers samples conditionally – only if they satisfy the given constraint.

In the context of reprocessing time modeling, a sample in SCR is the convergence length for a pair of states $L^k(s_i, s_j)$ on a piece of training input k . The constraint for the samples is the chunk size $\|I\| / n$. For a given state pair (s_i, s_j) , SCR maintains a short list of K samples⁶. However, during the regression analysis, SCR only chooses samples with convergence length shorter than the constraint (i.e., $L^k(s_i, s_j) < \|I\| / n$). Note that the constraint can vary, depending on the input size $\|I\|$ and the number of cores n . This flexibility allows the customization of SCR based on the needs of scalability analysis. On the other hand, with a pool of samples, the differences among samples resemble the variation of reprocessing costs among different chunks.

Note that convergence length profiling for different state pairs is already required by existing speculative FSM parallelization [37, 38], in order to improve the starting state prediction. In these cases, SCR does not require any extra profiling.

However, maintaining a list of samples for every pair of states could be expensive in terms of both space cost and the cost of regression analysis, depending on the number of high-frequency state pairs, their convergence lengths and the size of a sample set. To reduce the total amount of samples, SCR maintains samples only for state pairs that are more likely to appear in actual runs. To find out these state pairs, SCR performs a lightweight state pair frequency profiling offline, by invoking a speculative execution with a large number of parallel threads⁷. Let the set of high-frequency state pairs be S_f^2 , then

$$S_f^2 = \{(s, s') | \text{frequency}(s, s') > H_f\} \quad (10)$$

where H_f is a predefined frequency threshold.

With the samples of high-frequency state pairs, SCR computes the total reprocessing cost estimate T'_{repr} for a configuration n with the following equation:

$$T'_{repr}(n) = \frac{n-1}{\|S_f^2\| \cdot K} \sum_{(s, s') \in S_f^2} \sum_{k=1}^K L^k(s, s'), \text{ where} \quad (11)$$

$$L^k(s, s') = \begin{cases} L^k(s, s'), & L^k(s, s') < \|I\| / n \\ \|I\| / n, & \text{otherwise} \end{cases}$$

Note that each sample $L^k(s, s')$ is considered only if it satisfies the constraint, that is, $L^k(s, s') < \|I\| / n$. Statistically speaking, if set $H_f = 0$, then we have

$$\lim_{K \rightarrow \infty, n \rightarrow \infty} T'_{repr}(n) = T_{repr}(n) \quad (12)$$

Model M1. Putting all together, we have the estimated speculative execution time

$$T_{spec} = C_{pred} + \frac{\|I\|}{n} + T'_{repr}(n) \quad (13)$$

As the sequential execution time $T_{seq} = \|I\|$, we have the first scalability model M1:

$$S_{M1} = \frac{T_{seq}}{T_{spec}} = \frac{\|I\|}{C_{pred} + \|I\| / n + T'_{repr}(n)} \quad (14)$$

Based on Equation 14, for a given FSM and an input size $\|I\|$, Model M1 can compute the speedup of speculative FSM parallelization

⁶ K is tunable to balance the accuracy and cost. In our evaluation, K is set to 120.

⁷In our experiments, this number is set to 1000.

for any configuration n , with the help of SCR (Equation 11), hence, find out the optimal configuration n^* , such that,

$$S_{M1}(n^*) = \max\{S_{M1}(n) | 1 \leq n \leq N_{core}\} \quad (15)$$

Depending on the number of state pairs $\|S_f^2\|$ and the number of samples for each state pair K , the calculation of Equation 11 may introduce some runtime cost. One way to reduce the cost is by tuning the state pair frequency threshold H_f and the number of samples K , which in turn may compromise the accuracy. Next, we will discuss another way to balance the accuracy and modeling cost, by simplifying the SCR model. The simplification will lead to a closed-form representation of the optimal configuration.

Model M2. Considering the SCR model in Equation 11, there are two scenarios in which the model can be simplified by eliminating the constraint. First, when the input size is large enough or the convergence lengths between state pairs are relatively short, such that $L^k(s, s')$ is often smaller than $\|I\|/n$, then we can assume that $\mathcal{L}^k(s, s') = L^k(s, s')$. Second, when the speculative parallelization does not adopt the state convergence optimization (e.g., because few states can converge, like FSM div in [38]), or the convergence length is so long, such that $L^k(s, s')$ is often larger than the chunk size, then we assume $\mathcal{L}^k(s, s') = \|I\|/n$. Putting two scenarios together, we have a new model for reprocessing time estimation:

$$T''_{repr}(n) = \begin{cases} \bar{L} \cdot (n-1), & \bar{L} < \|I\|/n \\ (1 - P_s) \cdot (n-1) \cdot \|I\|/n, & \text{otherwise} \end{cases} \quad (16)$$

where $\bar{L}$ is the average convergence length among all samples and P_s is the probability of successful speculation, that is, $P(s = s')$. Depending on the ratio between input size and number of cores, the new model $T''_{repr}(n)$ switches between two equations.

By substituting the corresponding term in Equation 14 with the new model, we get the second scalability model M2:

$$S_{M2} = \frac{T_{seq}}{T_{spec}} = \frac{\|I\|}{C_{pred} + \|I\|/n + T''_{repr}(n)} \quad (17)$$

One advantage of Model M2 is that the optimal number of cores n^* can be represented in a closed-form expression, hence calculated directly without going through the pool of samples (required by Model M1). Considering Equation 16 and Equation 17 together, we can solve speedup maximization problem in Equation 3 and get the following optimal configuration:

$$n^* = \begin{cases} \sqrt{\|I\|/\bar{L}}, & \bar{L} < \|I\|/N_{core} \\ \sqrt{1/(1 - P_s)}, & \text{otherwise} \end{cases} \quad (18)$$

where $\bar{L}$ and P_s capture the convergence properties of the FSM and the speculation accuracy, respectively.

Equation 18 quantitatively reflects two basic intuitions behind scalability analysis. First, as the convergence length $\bar{L}$ increases, the optimal configuration n^* should be reduced. Second, when the speculation accuracy P_s increases, the speedup tends to be better when choosing to use more available cores.

Discussion. Comparing models M1 and M2, there is a tradeoff between the accuracy and the modeling cost. On one hand, with the SCR, Model M1 captures more details of misspeculation cost variation, hence tends to be more accurate in most cases. Meanwhile, M1 incurs more overhead as it needs to go through the pool of samples to calculate the speedup for each configuration. On the other hand, though Model M2 directly computes the optimal configuration, it

may lose some accuracy, especially when the average convergence length $\bar{L}$ is close to the chunk size $\|I\|/n$.

Both models M1 and M2 are solely based on FSMs' properties, and can be used for comparing the scalability of different FSMs when being executed speculatively. In practice, the actual scalability also depends on the characteristics of underlying architecture. Next, we will discuss how to extend the FSM properties-based scalability models to *architecture-aware scalability models*.

5 TOWARDS ARCHITECTURE-AWARE SCALABILITY ANALYSIS

On different architectures, the scalability of a type of computations may vary significantly, not only depending on the characteristics of the architecture, but also depending on their interaction with the computations. To enable accurate and practical scalability analysis for a given computing platform, this section presents architecture characterizations and discusses how to integrate them into the scalability models introduced in Section 4.

5.1 Architecture Effects

Considering the complexity of modern architectures, the interplay between an architecture and an application can be quite involved. Here, we focus on the end-to-end architecture effects that are closely relevant to the performance of speculative FSM parallelization. In another word, our architecture characterizations are customized for speculative FSM parallelization.

Since the execution time of speculative FSM execution mainly consists of two phases: parallel processing phase and sequential reprocessing phase (Section 4.2), we separate our discussion on the two phases. In specific, for each phase, we identify the major factor(s) that directly influences the performance.

Resource Contention in Parallel Phase. During the parallel phase, a group of n threads are created, each of them occupying a separate (logical) core. Based on their predicted starting states, these threads proceed with their own input chunks individually, and do not need to communicate either other. Thus, they do not suffer from any lock contention that is often caused by concurrent access of the shared data structures⁸. However, different cores physically share hardware resources, such as last level cache (LLC) and memory bandwidth, and even more resources among logical cores in a hyper-threaded core. The sharing of resources leads to contentions that directly influence the performance of this phase.

As the number of cores used increases, the resource contention tends to increase as well. However, the contention may not increase linearly or even monotonically, depending on the design of the architecture as well as the mapping between threads and logical cores. Without loss of generality, this work assumes that S3 uses the default mapping that is chosen by the operations system (defined in `/proc/cpuinfo`).

To quantitatively measure the resource contention, we introduce the metric *contention factor*, denoted as $\alpha(n)$.

$$\alpha(n) = T(n)/T(1) \quad (19)$$

where $T(i)$ is the execution time of processing i input chunks of the same length with i cores. Contention factor $\alpha(n)$ captures the degree of resource contention when executing with n parallel threads, comparing with a single thread execution. For commonly used architectures, it is expected that $\alpha(n) > 1$. For a given architecture,

⁸Threads do share the same FSM transition table, but only perform read operations.

the contention factor $\alpha(n)$ can be easily measured by running a micro benchmark N_{core} times.

Relative Execution Speed of Reprocessing. After entering into the reprocessing phase, only one thread is left, responsible for validating the correctness of each speculation and correcting the mistakenly processed parts caused by misspeculation. This implies there is no resource contention in this phase (i.e., $\alpha(1) = 1$). However, due to the tracking of state convergence (between s_{pred} and s_{true}), the execution speed (i.e., processing time per symbol) in reprocessing phase might be relatively slower than regular state transitions. The actual difference depends on the architecture, meanwhile, affects the scalability: the slower the reprocessing is, the less scalability the speculative parallelization can achieve.

To capture the relative execution speed, we introduce the *relative speed* factor, denoted as γ .

$$\gamma = T_{repr}^0 / T_{seq}^0 \quad (20)$$

where T_{repr}^0 and T_{seq}^0 are the processing time of a single symbol during reprocessing and a sequential execution, respectively. It is also expected that $\gamma > 1$. Similar to contention factor, γ can be measured with a micro FSM benchmark, but just running twice.

For a given architecture, $\alpha(n)$ and γ only need to be profiled once. Next, we discuss how to integrate these two architecture factors into the scalability models presented in Section 4. The integration will lead to a pair of architecture-aware scalability models that are more accurate and practical than their counterparts.

5.2 Integration of Architecture Factors

We first consider the resource contention factor $\alpha(n)$ in the parallel phase, then the relative speed factor γ in the reprocessing phase, and finally put them together.

The parallel phase execution time model in Equation 6 assumes that the parallel processing time T_{proc} equals sequential processing time (modeled as $\|I\|$) divided by the number of cores n . When considering the resource contention factor (Equation 19), that is, $\alpha(n) = T_{proc} / (T_{seq}/n)$, we can easily infer

$$T_{proc} = \alpha(n) \cdot \frac{T_{seq}}{n} = \alpha(n) \cdot \frac{\|I\|}{n} \quad (21)$$

Equation 21 implies that the higher the resource contention is, the longer the parallel processing time would become.

Similarly, we can extend to the reprocessing phase model in Equation 7 by integrating relative speed factor γ

$$T_{repr} = \gamma \cdot \sum_{i=2}^n L^i(s_{pred}^i, s_{true}^i) \quad (22)$$

Putting all together, we have two enhanced scalability models M1+ and M2+, corresponding to models M1 and M2, respectively.

Model M1+. Based on Model M1 in Equation 14, we have the following extended Model M1+ with architecture factors.

$$S_{M1+} = \frac{T_{seq}}{T_{spec}} = \frac{\|I\|}{C_{pred} + \alpha(n) \cdot \|I\| / n + \gamma \cdot T'_{repr}(n)} \quad (23)$$

Model M2+. Similarly, based on Equations 16 and 17, we extend Model M2 to Model M2+ as follows.

$$S_{M2+} = \frac{T_{seq}}{T_{spec}} = \frac{\|I\|}{C_{pred} + \alpha(n) \cdot \|I\| / n + \gamma \cdot T''_{repr}(n)} \quad (24)$$

where $T''_{repr}(n)$ is defined the same as that in Equation 16.

Augmented with the architecture factors, models M1+ and M2+ are expected to provide more accurate scalability analysis results that are customized to a specific architecture.

6 IMPLEMENTATION

We implemented S3 based on the OptSpec library [37, 38], which is implemented in C language and leverages Pthread for multi-threading. At high level, there are three major components: (i) An FSM property collector for profiling state convergence properties. The collector can be tuned either online using testing inputs or offline using training inputs. The cost of online profiling has been optimized with techniques from prior work [37] (typically less than 5%); (ii) An offline architecture property collector which runs a small set of micro FSM benchmarks on the target machine to measure the resource contention $\alpha(n)$ and relative speed factor γ ; And (iii) a runtime controller that implements the scalability models. Based on the collected the FSM and architectural properties, the controller calculates the optimal configuration n^* , and feeds it into speculative parallelization setting at runtime.

7 EVALUATION

In this section, we evaluate S3 on large-scale shared memory architectures, including a standalone Xeon Phi processor with 256 logical cores. The evaluation mainly focuses on two aspects: the accuracy of scalability analysis and the performance and energy benefits from using S3. We also discuss of scalabilities of some specific FSM computations based on experimental results.

7.1 Methodology

We compare S3 with two methods. One is the *default setting* of OptSpec [37, 38] which uses all available cores on the machine; The other is the *exhausted searching* that provides the ground truth of optimal configuration. In specific, given a FSM, the input size and an architecture with N_{core} cores, the exhausted searching executes the FSM with its inputs on the architecture using 1 to N_{core} cores to find the optimal number of cores. Obviously, it is unreasonable to use exhausted searching in real situations as trying one configuration is already at least as costly as the executing the best configuration, not mentioning enumerating all configurations.

We run our experiments on three different architectures, which are summarized in Table 1. Due to space limit, we mainly focus on the results on Xeon Phi architecture. Xeon Phi runs Linux 3.10.0 with GCC 4.8.5, while the other two run Linux 3.10.0 with GCC 4.47. All programs are compiled with “-O3” optimization flag. The timing results reported are the average of 10 runs on 10 inputs, with all runtime cost included. We use PAPI [3] for accessing hardware performance counters to measure the energy consumption.

Table 1: Architectures in Evaluation

Arch.	model	freq.	#cores*	#SMT	#Sockets
Xeon Phi	Xeon Phi 7210	1.30 Hz	256	4	1
Haswell	Xeon E5-2698	2.30 Hz	64	2	2
Ivy Bridge	Xeon E7-8860	2.27 Hz	80	1	8

*The numbers of cores shown are the number of logical cores recognized by operating systems.

The benchmarks are collected from real-world FSM applications, primarily from Snort [32], one of the most widely used open source Network Intrusion Detection Systems (NIDSs). It has a rich body of signatures/rules, most of which are specified by a Perl-compatible regular expression (PCRE). We converted the PCREs to FSMs using

Table 2: FSM Benchmarks and Their Properties

FSM	#states	avg(L)	FSM	#states	avg(L)
openview	501	1.27E+07	spirit	2041	8.09E+01
tomcat	10	6.56E+06	jnl	1211	8.09E+01
iis	32	1.27E+07	postgre	28	9.98E+04
cnc	115	9.93E+04	apache	6	2.55E+05
rtf	768	8.66E+04	mutiny	21	2.53E+05
warehouse	82	1.81E+06	buffer	10	1.12E+06
dfs	26	6.83E+04	adware	5265	1.20E+02

standard regular expression to FSM conversion algorithms [1]. The inputs to the FSMs are network traffic traces collected from a Linux server and a laptop via tcpdump, with a total size of 18GB. Table 2 summarizes the 14 benchmarks used in our evaluation, including the number of states and the average convergence length collected from high-frequency state pairs, each with 120 samples.

7.2 Model Accuracy

Table 3 reports the optimal configuration n^* found by exhausted searching and the four models of S3, on architectures Xeon Phi and Haswell⁹. “Exs” shows the actual optimal number of cores by enumerating all configurations (i.e., the “ground truth”). Note that, the predicted optimal number of cores is bounded by the total number of cores in the tested platforms. Overall, architecture-aware models (M1+ and M2+) are more accurate than architecture-independent models (M1 and M2), especially for benchmarks with better scalabilities, thanks to their consideration of architecture factors $\alpha(n)$ and γ . The differences between M1+ and M2+ are not significant for most benchmarks, similar to M1 and M2. The largest difference happens on benchmark *rtf*, where M2 turns to be much over optimistic (80 v.s. 42). In comparison, the result of M1 is very close to the actual optimal (43 v.s. 42). Also note that the results of M2 is closer to the ground truth than M1 in general. On one hand, due to the simplification of SCR model, M2 predicts less accurately than M1 in reprocessing length (more pessimistic in most cases). On the other hand, both M1 and M2 miss the architecture factors as mentioned above, and tend to be more optimistic. Because of the “balance” that happens to M2, results from M2 turn to be closer to the real cases than M1.

Table 3: Optimal Configurations from Different Methods

	Xeon Phi					Haswell				
	Exs	M1	M1+	M2	M2+	Exs	M1	M1+	M2	M2+
openview	6	+1	0	+1	0	7	0	0	0	0
tomcat	8	+1	0	0	0	12	-3	-1	-3	-1
iis	6	+1	0	+1	0	7	0	0	0	-1
warehouse	15	+3	0	+3	0	21	-3	+1	-3	+1
apache	37	+11	+3	+10	0	53	-5	-4	-6	-4
mutiny	34	+15	+5	+13	+3	44	+5	+3	+3	+3
buffer	26	+7	+1	-4	-9	34	-1	+6	-12	-12
cnc	51	+32	+9	+24	+3	62	0	-3	+2	-3
rtf	42	+1	-1	+38	+21	54	-11	-5	+10	+8
dfs	54	+46	+8	+36	+8	59	+5	-1	+5	-2
postgre	50	+35	+7	+24	+7	56	+7	-1	+8	-1
adware	110	+146	+2	+146	+2	58	+6	-1	+6	-1
spirit	116	+140	0	+140	0	62	+2	-4	+2	-2
jnl	116	+140	-2	+140	-2	60	+4	-1	+4	-1

To further examine the overall accuracy in scalability analysis, we collected the speedup curves for each benchmark on each architecture. Due to space limit, we only report some representative

⁹The results on Ivy Bridge follow similar patterns.

results in Figure 5. In general, the speedup curves clearly show the effectiveness of the two architectural-aware models (M1+ and M2+), whose speedup curves precisely align with the actual speedup curve for most benchmarks. Between M1+ and M2+, M2+ performs less reliable than M1+, especially on benchmarks *openview* and *buffer*, due to its simplification of the SCR model (Section 4). Model M1+ shows some slight discrepancy on benchmarks *buffer*, *openview*, and *mutiny*. This is mainly caused by the characteristic differences between the samples and the testing inputs.

7.3 Performance Improvement

We next present the performance benefits of S3, comparing with the default setting of speculative FSM parallelization [37, 38]. Figure 6 shows the speedup (baseline is sequential FSM execution) of all five methods on 14 benchmarks and three architectures. “Exhaust” represents the ideal speedup that can be achieved by tuning the number of cores. The most performance gains come from the results on Xeon Phi, for its larger number of available cores. On average, S3 boosts the speedup from 6.1X to 16.7X with Model M1+. For architecture-independent models (M1 and M2), the improvements are slightly less, but still reaching 15X. For benchmark *buffer*, the speedup is improved by a factor of five (3.2X v.s. 15.6X). Results on Haswell and Ivy Bridge follow similar trends in general, but are less significant due their limited number of available cores. Overall, the results imply the necessity of scalability-aware speculative FSM parallelization, especially considering future parallel platforms with even more number of processing units.

Table 4: Energy Saving by S3 (baseline: Default Setting)

	Xeon Phi				Haswell			
	M1	M1+	M2	M2+	M1	M1+	M2	M2+
openview	67.7%	67.1%	67.7%	67.1%	37.2%	37.2%	37.2%	37.2%
tomcat	78.6%	75.7%	75.7%	75.7%	46.0%	44.0%	46.0%	44.0%
iis	69.5%	67.7%	69.5%	67.7%	41.2%	41.2%	41.2%	40.4%
warehouse	79.9%	80.3%	79.9%	80.3%	18.7%	19.6%	18.7%	19.6%
apache	67.2%	68.0%	67.1%	68.3%	-1.4%	-1.8%	-2.4%	-1.8%
mutiny	68.0%	69.2%	69.6%	69.5%	-0.8%	-0.6%	-0.6%	-0.6%
buffer	76.9%	77.3%	77.2%	75.3%	-11.4%	-1.4%	-18.9%	-18.9%
cnc	49.7%	54.8%	51.8%	55.8%	0.3%	1.2%	0.0%	1.2%
rtf	66.3%	66.3%	57.2%	64.1%	-8.9%	-4.7%	0.0%	0.7%
dfs	35.4%	48.1%	37.6%	48.1%	0.0%	5.0%	0.0%	5.0%
postgre	47.3%	54.5%	50.9%	54.5%	-1.6%	1.0%	0.0%	1.0%
adware	0.0%	13.7%	0.0%	13.7%	0.0%	3.6%	0.0%	3.6%
spirit	0.0%	15.2%	0.0%	15.2%	0.0%	3.9%	0.0%	3.9%
jnl	0.0%	12.9%	0.0%	12.9%	0.0%	4.8%	0.0%	4.8%

7.4 Energy Saving

Finally, we briefly discuss one side benefit of scalability-sensitive speculative FSM parallelization – energy saving. The energy saving primarily comes from the use of less number of processors. Table 4 reports the energy saving in percentage on Xeon Phi and Haswell architectures. On Xeon Phi, the energy saving is more significant, up to 77%, because it has more room to reduce the amount of core uses. However, on Haswell, we also observed cases with even more energy consumption. This is because when using all available cores (64 cores), though the power consumption is higher, the execution time becomes shorter (due to smaller chunk size $\|I\|/n$), comparing with the optimal core predicted by the models (e.g., 22 cores for M2 and M2+). In another word, for speculative FSM parallelization, it is not necessary that more number of cores always leads to higher energy consumption. We leave further investigation as future work.

Figure 5: Comparison of Speedup Curves from Different Models. (x-axis is the number of cores; y-axis is the speedup)

Figure 6: Speedup Comparison between S3 and Prior Work [38] on Three Architectures: Xeon Phi, Haswell, and Ivy Bridge

8 RELATED WORK

Scalability Analysis. Scalability analysis is critical for studying the performance of parallel applications, especially on large scale parallel computers. It has been used to find the best algorithm-architecture combinations for problems with different constraints and predict the performance of a parallel algorithm for a parallel platform with more number of processors [2, 10, 17]. The most widely used scalability models include Amdahl’s law and its variations [2], which states the existence of a speedup bound caused by the serial part. Gustafson further builds models whose system resources are improved, known as Gustafson’s law [8]. Though providing valuable insights, these scalability models cannot be applied to speculative FSM parallelization whose behaviors are probabilistic with non-linear variation of parallelization cost.

Speculative FSM Parallelization. Due to the dependencies in state transitions, existing ways to parallelize FSM rely on either enumeration-based parallel prefix-sum and its variations [18, 22], or speculative parallelization [27, 37, 38]. The former can be treated as a special case of speculative parallelization, where the “speculation” enumerates all the states, hence always covers the correct one. From this perspective, though the models proposed in this work can be reused for the former with simple extensions.

Some other FSM parallelization work focus on a few specific FSM applications, such as browser front-end [11] and JPEG decoder [13]. The basic ideas in these work were later formalized by Zhao and others [38] by introducing a concept called *principled speculation*. Other examples include hot state prediction for FSMs in intrusion detection [21] and speculative parsing [12]. Some studies also design and implement parallel Non-deterministic Finite Automata (NFA) [39], which naturally exposes parallelism, hence are relatively easier to parallelize, comparing with their DFA counterparts.

Some prior work have also explored bit-parallel fine-grained parallelism for FSMs by converting FSM computations into a sequence of bit operations [19, 23], and the combination of both fine-grained and coarse-grained speculative parallelism [27].

Speculative Parallelization of Other Applications. The idea of speculative parallelization has been studied for years. These work include designing new programming language constructs [26] and developing parallelization frameworks [5, 7, 29, 30, 35]. Some of these studies have explored parallelism in irregular programs [9, 15, 20, 25, 31], which share some similarities with the parallelization of FSM computations, given that FSMs essentially run on an irregular data structure (a graph). Quinones and others [28] use pre-computation for speculative threading, which shares the idea with speculative FSM parallelization in that both exploit some constraints of the computation to facilitate speculative execution.

9 CONCLUSION

With a systematic scalability study, this work points out a principal fallacy in the existing design of speculative FSM parallelization when being ported to a larger parallel platform. To address the issue, this work introduces a series of scalability analysis models, which are tailored to both the properties of FSM computations and the characteristics of the underlying architecture. To leverage the proposed models, this work develops an automatic speculative FSM parallelization framework S3, which, for the first time, enables a scalability-sensitive speculative parallelization for FSM computations. For a given FSM, its input size and the architecture, S3 can automatically compute the optimal number of cores to use and guide the speculative parallelization towards the best performance. Evaluation on FSM benchmarks with a spectrum of scalabilities demonstrates the effectiveness of the new speculative parallelization scheme, showing up to 5X speedup comparing to previous methods as well as up to 77% energy saving.

ACKNOWLEDGMENTS

We thank all anonymous reviewers for their constructive comments. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 1565928.

REFERENCES

- [1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. 1986. *Compilers, Principles, Techniques*. Addison wesley Boston.
- [2] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 483–485.
- [3] S. Browne, C. Deane, G. Ho, and P. Mucci. 1999. PAPI: A portable interface to hardware performance counters. In *Proceedings of Department of Defense HPCMP Users Group Conference*.
- [4] Sutapa Datta and Subhasis Mukhopadhyay. 2015. A grammar inference approach for predicting kinase specific phosphorylation sites. *PLoS one* 10, 4 (2015), e0122294.
- [5] C. Ding, X. Shen, K. Kelsey, C. Tice, R. Huang, and C. Zhang. 2007. Software Behavior-oriented Parallelization. In *PLDI*.
- [6] Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. 2014. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems* 25, 12 (2014), 3088–3098.
- [7] M. Feng, R. Gupta, and Y. Hu. 2011. SpiceC: Scalable parallelism via implicit copying and explicit commit. In *Proceedings of the ACM SIGPLAN Symposium on Principles Practice of Parallel Programming*.
- [8] John L Gustafson. 1988. Reevaluating Amdahl's law. *Commun. ACM* 31, 5 (1988), 532–533.
- [9] Maurice Herlihy and Eric Koskinen. 2008. Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '08)*.
- [10] Adolfo Hoisie, Olaf Lubeck, and Harvey Wasserman. 2000. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *The International Journal of High Performance Computing Applications* 14, 4 (2000), 330–346.
- [11] C. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik. 2009. Parallelizing the web browser. In *HotPar*.
- [12] B. Kaplan. Speculative parsing path. (????). <http://bugzilla.mozilla.org>.
- [13] S. Klein and Y. Wiseman. 2003. Parallel huffman decoding with applications to jpeg files. *Journal of Computing* 46, 5 (2003).
- [14] Shmuel Tomi Klein and Yair Wiseman. 2003. Parallel Huffman decoding with applications to JPEG files. *Comput. J.* 46, 5 (2003), 487–497.
- [15] Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala, and L. Paul Chew. 2007. Optimistic Parallelism Requires Abstractions. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07)*.
- [16] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. 2006. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *ACM SIGCOMM Computer Communication Review*, Vol. 36. ACM, 339–350.
- [17] Vipin Kumar and Anshul Gupta. 1991. Analysis of Scalability of Parallel Algorithms and Architectures: A Survey. In *Proceedings of the 5th International Conference on Supercomputing (ICS '91)*. ACM, New York, NY, USA, 396–405. <https://doi.org/10.1145/109025.109118>
- [18] Richard E. Ladner and Michael J. Fischer. 1980. Parallel Prefix Computation. *J. ACM* 27, 4 (Oct. 1980), 831–838.
- [19] Dan Lin, Nigel Medforth, Kenneth S Herdy, Arrivindh Shriraman, and Rob Cameron. 2012. Parabix: Boosting the efficiency of text processing on commodity processors. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 1–12.
- [20] Weifeng Liu and Brian Vinter. 2015. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Comput.* 49 (2015), 179–193.
- [21] D. Luchaup, R. Smith, C. Estan, and S. Jha. 2009. Multi-byte regular expression matching with speculation. In *RAID*.
- [22] Todd Mytkowicz, Madanlal Musuvathi, and Wolfram Schulte. 2014. Data-Parallel Finite-State Machines. In *ASPLOS '14: Proceedings of 19th International Conference on Architecture Support for Programming Languages and Operating Systems*. ACM Press.
- [23] Gonzalo Navarro. 2001. NR-grep: a fast and flexible pattern-matching tool. *Software: Practice and Experience* 31, 13 (2001), 1265–1312.
- [24] Yinfei Pan, Ying Zhang, Kenneth Chiu, and Wei Lu. 2007. Parallel xml parsing using meta-dfas. In *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 237–244.
- [25] Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M. Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui. 2011. The Tao of Parallelism in Algorithms. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)*.
- [26] P. Prabhu, G. Ramalingam, and K. Vaswani. 2010. Safe Programmable Speculative Parallelism. In *Proceedings of ACM SIGPLAN Conference on Programming Languages Design and Implementation*.
- [27] Junqiao Qiu, Zhijia Zhao, and Bin Ren. 2016. MicroSpec: Speculation-centric fine-grained parallelization for FSM computations. In *Parallel Architecture and Compilation Techniques (PACT), 2016 International Conference on*. IEEE, 221–233.
- [28] C.G. Quinones, C. Madriles, J. Sanchez, P. Marcuello, A. Gonzalez, and D. M. Tullens. 2005. Mitosis compiler: an infrastructure for speculative threading based on pre-computation slices. In *PLDI*.
- [29] A. Raman, H. Kim, T. R. Mason, T. B. Jablin, and D. I. August. 2010. Speculative parallelization using software multi-threaded transactions. In *Proceedings of the international conference on Architectural support for programming languages and operating systems*.
- [30] Lawrence Rauchwerger and David A. Padua. 1995. The LRPD Test: Speculative Run-Time Parallelization of Loops with Privatization and Reduction Parallelization. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI), La Jolla, California, USA, June 18-21, 1995*. 218–232.
- [31] Bin Ren, Gagan Agrawal, James R Larus, Todd Mytkowicz, Tomi Poutanen, and Wolfram Schulte. 2013. SIMD parallelization of applications that traverse irregular data structures. In *Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on*. IEEE, 1–10.
- [32] Martin Roesch et al. 1999. Snort: Lightweight Intrusion Detection for Networks.. In *LISA*, Vol. 99. 229–238.
- [33] Indranil Roy and Srinivas Aluru. 2014. Finding motifs in biological sequences using the micron automata processor. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 415–424.
- [34] Priti Shankar, Amitava Dasgupta, Kaustubh Deshmukh, and B Sundar Rajan. 2003. On viewing block codes as finite automata. *Theoretical Computer Science* 290, 3 (2003), 1775–1797.
- [35] C. Tian, M. Feng, V. Nagarajan, and R. Gupta. 2008. Copy Or Discard Execution Model For Speculative Parallelization On Multicores. In *Proceedings of the International Symposium on Microarchitecture*.
- [36] Fang Yu, Zhifeng Chen, Yanlei Diao, TV Lakshman, and Randy H Katz. 2006. Fast and memory-efficient regular expression matching for deep packet inspection. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*. ACM, 93–102.
- [37] Zhijia Zhao and Xipeng Shen. 2015. On-the-Fly Principled Speculation for FSM Parallelization. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*. 619–630. <https://doi.org/10.1145/2694344.2694369>
- [38] Zhijia Zhao, Bo Wu, and Xipeng Shen. 2014. Challenging the "Embarrassingly Sequential": Parallelizing Finite State Machine-Based Computations through Principled Speculation. In *ASPLOS '14: Proceedings of 19th International Conference on Architecture Support for Programming Languages and Operating Systems*. ACM Press.
- [39] Y. Zu, M. Yang, Z. Xu, L. Wang, X. Tian, K. Peng, and Q. Dong. 2009. GPU-based NFA implementation for memory efficient high speed regular expression matching. In *PPoPP '12: Proceedings of the ACM SIGPLAN symposium on Principles and practice of parallel programming*. 129–140.