Report “ A Survey of Computer Architecture Simulation Techniques and Tools”

Abstract :

Trình mô phỏng kiến trúc máy tính đóng vai trò quan trọng trong việc thúc đẩy kiến trúc máy tính nghiên cứu. Với hướng nghiên cứu rộng hơn và số lượng mô phỏng được phát triển ngày càng tăng, việc chọn một trình mô phỏng cụ thể để sử dụng trở nên khó khăn hơn. Bài viết này xem xét các nguyên tắc cơ bản của các kỹ thuật mô phỏng kiến trúc máy tính. Nó cũng khảo sát nhiều mô phỏng kiến trúc máy tính và phân loại chúng thành các nhóm khác nhau dựa trên mô hình mô phỏng của chúng. So sánh kiến trúc máy tính các trình mô phỏng với nhau và xác nhận tính chính xác của chúng là những nhiệm vụ đòi hỏi khắt khe đối với các kiến trúc sư. Ngoài ra để cung cấp một cuộc khảo sát về các công cụ mô phỏng kiến trúc máy tính, chúng tôi đã đo được sai số thử nghiệm của sáu trình mô phỏng kiến trúc máy tính hiện đại: gem5, MARSSx86, Multi2Sim, PTLsim, Sniper và ZSim. Chúng tôi cũng đã thực hiện so sánh chi tiết các trình mô phỏng này dựa trên các tính năng khác như tính linh hoạt và các chi tiết kiến trúc vi mô. Chúng tôi tin rằng bài viết này sẽ là nguồn tài liệu rất hữu ích cho máy tính cộng đồng kiến trúc, đặc biệt dành cho các nhà nghiên cứu hệ thống và kiến trúc máy tính ở giai đoạn đầu để đạt được tiếp xúc với các tùy chọn mô phỏng kiến trúc hiện có.

Index Terms : Trình mô phỏng kiến trúc máy tính, kỹ thuật mô phỏng, xác nhận, trình mô phỏng x86,đánh giá trình mô phỏng.

I . INTRODUCTION

Các kiến trúc sư máy tính sử dụng mô phỏng để đánh giá các phương án thiết kế khác nhau, thử nghiệm các ý tưởng nghiên cứu mới và phân tích hiệu suất/mức tiêu thụ năng lượng của các mẫu bộ xử lý khác nhau. Các mô hình phân tích không phù hợp để đánh giá các thiết kế kiến trúc/vi kiến trúc và các biến thể thiết kế vì chúng tạo ra kết quả không chính xác do số lượng lớn cấu hình và các chi tiết nhỏ có thể gây ra những thay đổi nhỏ về hiệu suất. Mô phỏng được coi là phương pháp mô hình hóa hiệu suất tiêu chuẩn [1]. Phần lớn nghiên cứu được công bố đều dựa trên việc sử dụng mô phỏng để phân tích hiệu quả của các ý tưởng mới. Nhiều trình mô phỏng kiến trúc máy tính hỗ trợ nhiều kiến trúc tập lệnh (ISA), vi kiến trúc khác nhau và dựa trên các mô hình mô phỏng khác nhau, từ dựa trên dấu vết đến chính xác theo chu kỳ. Việc chọn một trình mô phỏng và bắt đầu nghiên cứu của họ có thể là một nhiệm vụ khó khăn đối với các nhà nghiên cứu mới trong lĩnh vực kiến trúc và hệ thống máy tính. Ngoài ra, để có được niềm tin vào các nghiên cứu mô phỏng, kết quả mô phỏng cần phải được xác thực. Điều này có thể là một thách thức, đặc biệt khi không có đủ tài liệu về trình mô phỏng. Có rất ít công việc đánh giá các trình mô phỏng kiến trúc máy tính hiện tại, thực hiện so sánh giữa chúng và/hoặc so sánh độ chính xác của chúng với các bộ xử lý hiện đại. Đóng góp lớn của chúng tôi trong bài viết này là :

* Cung cấp bản khảo sát cập nhật về các kỹ thuật mô phỏng và mô phỏng kiến trúc máy tính.
* Phân loại, phân tích và so sánh các trình mô phỏng kiến trúc máy tính khác nhau, có thể giúp cộng đồng hiểu được trường hợp sử dụng của các công cụ mô phỏng khác nhau.
* Cung cấp các đặc tính chi tiết và so sánh lỗi thử nghiệm của sáu trình mô phỏng kiến trúc máy tính x86 hiện đại: gem5 [2], Multi2sim [3], MARSSx86 [4], PTLsim [5], Sniper [6] và ZSim [7].
* Xem xét những thách thức quan trọng nhất đối với trình mô phỏng kiến trúc và các giải pháp đã được đề xuất để giải quyết những vấn đề đó.

Trong kiến trúc máy tính, mục tiêu chính của mô phỏng là

để mô hình hóa các ý tưởng nghiên cứu mới cho các bộ phận của hệ thống máy tính (ví dụ: bộ vi xử lý, bộ nhớ, thiết bị IO) hoặc hệ thống máy tính hoàn chỉnh và ước tính mức độ cải thiện hiệu suất và/hoặc mức tiêu thụ điện năng. Trình mô phỏng cũng giúp các kiến trúc sư máy tính đánh giá, gỡ lỗi và hiểu hành vi của các hệ thống hiện có. Trong thuật ngữ mô phỏng, hệ thống máy tính đang được mô phỏng được gọi là mục tiêu và hệ thống nơi chạy mô phỏng được gọi là máy chủ. Khối lượng công việc chạy trên trình mô phỏng có thể là các chương trình kiểm tra tiêu chuẩn, được gọi là điểm chuẩn, được chạy để đánh giá hiệu suất của bộ xử lý hoặc máy tính. Khối lượng công việc được mô phỏng cũng có thể là một hệ điều hành (OS), đôi khi được gọi là HĐH khách. Sự tương tác giữa trình mô phỏng và hệ thống máy chủ được thể hiện trong Hình 1. Khảo sát này là một khảo sát toàn diện hơn và cập nhật hơn so với các khảo sát hiện có trước đó, tập trung vào việc giảng dạy các trình mô phỏng liên quan [8]–[11], mô phỏng bộ nhớ [12 ], [13] hoặc không có nhiều chi tiết. Nikolic và cộng sự. [10] đã khảo sát nhiều mô phỏng kiến trúc máy tính phù hợp cho việc giảng dạy các khóa học kiến trúc máy tính. Họ đánh giá các mô phỏng khác nhau dựa trên tiêu chí của các chủ đề được đề cập trong lớp học và các tính năng mô phỏng. Uhlig và Mudge [12] và Holliday [13] đã thảo luận về các kỹ thuật mô phỏng bộ nhớ khác nhau, sử dụng dấu vết địa chỉ tham chiếu. Urden [14] đã so sánh và đánh giá kết quả hoạt động của ba trình mô phỏng kiến trúc máy tính khác nhau với nhau. Tuy nhiên, nghiên cứu của ông không so sánh mô phỏng với hoạt động phần cứng thực. Trong khảo sát này, chúng tôi so sánh kết quả của trình mô phỏng với kết quả của các thử nghiệm phần cứng thực tế để đo lường độ chính xác của chúng. Nowatzki và cộng sự. [15] đã thảo luận về những cạm bẫy khác nhau liên quan đến việc sử dụng các mô phỏng kiến trúc. Họ cũng đã thảo luận về các lỗi mà họ quan sát thấy trong bốn trình mô phỏng hiệu suất và sức mạnh: gem5 [2], GPGPUSim [16], McPAT [17] và GPUWattch [18]. Các nỗ lực xác thực cho các trình mô phỏng khác nhau (ví dụ: SimpleScalar, SMARTS, Microlib, gem5, Sniper, SiNUCA, Ramulator) cũng đã được xuất bản [19]–[25]. Các bài viết này chỉ tập trung vào một trình mô phỏng đang được xác thực và thường không bao gồm các so sánh với các công cụ liên quan. Khảo sát này được cập nhật, bao gồm các trình mô phỏng kiến trúc bộ xử lý mới hơn và chi tiết hơn so với các khảo sát trước đó. Cuộc khảo sát cũng so sánh và đối chiếu một số mô phỏng hiện đại. Để đẩy nhanh quá trình mô phỏng, các nhà nghiên cứu thường dựa vào kỹ thuật lấy mẫu. Bài viết cũng thảo luận về một số phương pháp lấy mẫu thường được sử dụng liên quan đến việc mô phỏng các thành phần kiến trúc máy tính. Tương tự, độ chính xác của kết quả mô phỏng luôn là mối quan tâm của các kiến trúc sư. Chúng tôi khám phá ngắn gọn các phương pháp xác thực trình mô phỏng khác nhau hiện có. Hơn nữa, công việc này so sánh lỗi thử nghiệm của sáu trình mô phỏng x86 với các lần chạy phần cứng và cung cấp hiệu suất tương đối của các trình mô phỏng khi thay đổi một số cấu hình vi kiến trúc. Cuối cùng, chúng tôi xác định một số nguyên nhân dẫn đến sự thiếu chính xác trong kết quả mô phỏng mà chúng tôi quan sát được.

Phạm vi của bài viết: Bởi vì nghiên cứu kiến trúc máy tính bao gồm nhiều loại kiến trúc, từ vi kiến trúc bộ xử lý đến các kiến trúc và máy gia tốc có mục đích đặc biệt, sử dụng các loại mô phỏng khác nhau, nên chúng tôi giới hạn phạm vi của bài viết để xem xét chi tiết các mô phỏng kiến trúc và vi kiến trúc máy tính và mô hình mô phỏng bộ vi xử lý. Bài viết đề cập đến một số trình mô phỏng uncore và accelerator. Nó không khảo sát chi tiết các mô phỏng chuyên dụng độc lập cho các cấu trúc kiến trúc kỳ lạ hoặc các thành phần không có lõi; tuy nhiên, nó có đề cập đến các mô phỏng vi kiến trúc hoặc không cốt lõi chi tiết như các tính năng của các trình mô phỏng kiến trúc/vi kiến trúc máy tính đầy đủ đó. Nó cũng thảo luận về những bộ mô phỏng đầy đủ cũng có thể mô phỏng máy gia tốc hoặc có thể được gắn vào bộ mô phỏng máy gia tốc. Bài viết không đề cập đến các hệ thống mô phỏng trên chip (SoC), mặc dù nhiều trình mô phỏng được thảo luận có khả năng mô phỏng các bộ xử lý nhúng. Bài viết cũng thảo luận về các phương pháp mô phỏng khác nhau, phân loại chúng, so sánh và đối chiếu các kỹ thuật đó. Ngoài ra, bài viết thảo luận về những thách thức liên quan đến mô phỏng và các giải pháp khả thi, đồng thời thảo luận về các kỹ thuật đánh giá mô phỏng. Tuy nhiên, để giới hạn phạm vi, bài viết không trình bày chi tiết các hạn chế triển khai cụ thể, chẳng hạn như xử lý đa luồng mục tiêu.

Cấu trúc phần còn lại của bài báo như sau: Phần II phân loại các trình mô phỏng thành các loại khác nhau và thảo luận chi tiết về các loại này. Phần IV tóm tắt các mô phỏng kiến trúc/vi kiến trúc máy tính khác nhau hiện có. Phần V khám phá những thách thức phải đối mặt trong mô phỏng kiến trúc máy tính và giải pháp của chúng. Phần VI thảo luận ngắn gọn về việc xác nhận tính hợp lệ của trình mô phỏng. Phần VII mô tả chi tiết sáu trình mô phỏng x86 hiện đại mà chúng tôi đã chọn để đánh giá chi tiết. Phần VIII thảo luận về phương pháp được sử dụng để đo lường sai số thử nghiệm của các thiết bị mô phỏng và hiệu suất tương đối của chúng. Phần IX trình bày kết quả đánh giá của các trình mô phỏng x86. Cuối cùng, chúng tôi kết luận bài viết ở phần X.

II. CLASSIFICATION OF SIMULATORS

Trình mô phỏng có thể được phân loại thành nhiều nhóm khác nhau dựa trên ba yếu tố quan trọng nhất: chi tiết mô phỏng, phạm vi mục tiêu và đầu vào của trình mô phỏng. Phần này thảo luận chi tiết về phân loại phân loại đã nói ở trên. Cần lưu ý rằng sự phân loại này không loại trừ lẫn nhau và một trình mô phỏng có thể thuộc nhiều lớp. Ngoài ra, một số trình mô phỏng được phân loại dựa trên các khía cạnh hoặc chuyên môn nhất định cũng được thảo luận trong phần này.

A. PHÂN LOẠI MÔ PHỎNG DỰA TRÊN CHI TIẾT MÔ PHỎNG

Một yếu tố quan trọng để phân loại trình mô phỏng là mức độ chi tiết mà bất kỳ trình mô phỏng nào thực hiện trong thiết kế của nó. Các lớp mô phỏng chính dựa trên chi tiết mô phỏng là mô phỏng chức năng, thời gian và chức năng/thời gian.

1) MÔ PHỎNG CHỨC NĂNG

Trình mô phỏng chức năng chỉ triển khai kiến trúc và tập trung vào việc đạt được chức năng tương tự của kiến trúc được mô hình hóa. Nói cách khác, trình mô phỏng chức năng hoạt động giống như trình mô phỏng (mô phỏng hành vi của kiến trúc tập lệnh (ISA) của mục tiêu). Chúng thường nhanh hơn các loại trình mô phỏng khác, nhưng chúng không thể theo dõi các tham số vi kiến trúc chi tiết khi chương trình chạy trên trình mô phỏng vì chúng không triển khai vi kiến trúc. Trong khi phát triển các tập lệnh mới, các trình mô phỏng chức năng có thể được sử dụng cho mục đích thử nghiệm. Hơn nữa, trình mô phỏng chức năng có thể giúp xác định các đặc điểm kiến trúc của việc thực thi chương trình, chẳng hạn như tổng số loại lệnh khác nhau trong chương trình, vùng truy cập bộ nhớ, v.v. Hình 2(a) hiển thị sơ đồ khối của trình mô phỏng chức năng.

SimpleScalar [27] đã được sử dụng cho mục đích giảng dạy và nghiên cứu. SimpleScalar là một bộ công cụ toàn diện. Nó có nhiều mô hình mô phỏng khác nhau, trong đó *sim-safe* là một ví dụ về mô hình mô phỏng chức năng. Nó là một trình mô phỏng SimpleScalar tối thiểu chỉ mô phỏng ISA. Một phiên bản tối ưu hóa tốc độ của *sim-safe* được đặt tên là *sim -fast* [27]. Simics [28] là một trình mô phỏng chức năng khác, có khả năng độc đáo là thực hiện chương trình theo hướng tiến hoặc lùi. SimCore [29] là một trình mô phỏng chức năng cho bộ xử lý Alpha. Nó được cho là nhanh hơn 19% so với *simfast* của bộ công cụ SimpleScalar [29]. EduMIPS64 [30], một trình mô phỏng chức năng trực quan được viết bằng java cho MIPS, được thiết kế để sử dụng trong các lớp học dạy các khóa học kiến trúc máy tính. HASE [31] là một công cụ mô phỏng và hiển thị mức độ cao của kiến trúc máy tính. Nó được phát triển vào những năm 90 bằng cách sử dụng các ngôn ngữ mô phỏng hướng đối tượng. Dự án HASE cung cấp nhiều mô hình kiến trúc máy tính hướng tới giảng dạy các khái niệm liên quan đến kiến trúc máy tính. Barra [32] là một trình mô phỏng chức năng cho GPGPU (đơn vị xử lý đồ họa đa năng). Nó hỗ trợ mô phỏng các ứng dụng CUDA. Một ví dụ khác về mô phỏng chức năng là mô hình CPU ' AtomicSimple ' của *gem5* . Một cách khác để tạo một trình mô phỏng chức năng là cài đặt mã nhị phân của chương trình với mã chịu trách nhiệm thu thập thông tin cần thiết khi chương trình thực thi trên phần cứng thực [33]. Những công cụ như vậy được gọi là công cụ đo lường nhị phân động, ví dụ như công cụ Pin [34]. Có nhiều trình mô phỏng (ví dụ CMP$im [35], Sniper [6]), cũng dựa vào các công cụ thiết bị để thực hiện mô phỏng chức năng.

2) MÔ PHỎNG THỜI GIAN

Trình mô phỏng thời gian, còn được gọi là trình mô phỏng hiệu suất, mô phỏng vi kiến trúc của bộ xử lý (Hình 2(b)). Họ đưa ra số liệu thống kê chi tiết về thời gian/hiệu suất của hệ thống mục tiêu [26]. Ví dụ: trong trường hợp mô phỏng bộ xử lý, thông tin này có thể bao gồm các số liệu thống kê như hướng dẫn trên mỗi chu kỳ (IPC), thời gian chạy chương trình, hiệu suất của hệ thống bộ nhớ và các số liệu thống kê chi tiết khác liên quan đến kiến trúc vi mô. Trình mô phỏng thời gian/hiệu suất không bắt buộc phải mô phỏng chức năng của mục tiêu. Trình mô phỏng thời gian có các loại phụ khác nhau, tùy thuộc vào mức độ chi tiết có trong trình mô phỏng: trình mô phỏng cấp chu kỳ, trình mô phỏng theo hướng sự kiện và trình mô phỏng khoảng thời gian.

***Trình mô phỏng cấp độ chu kỳ:*** Trình mô phỏng cấp độ chu trình mô phỏng một kiến trúc bằng cách bắt chước hoạt động của bộ xử lý được mô phỏng cho mỗi chu kỳ. Ngược lại với các trình mô phỏng chính xác theo chu kỳ mô phỏng *chính xác* những gì xảy ra trên mỗi chu kỳ bằng cách sử dụng triển khai RTL [36], các trình mô phỏng ở cấp độ chu kỳ không mô hình hóa phần cứng với các chi tiết nhỏ. Trình mô phỏng cấp chu trình hoạt động chậm và sử dụng lượng bộ nhớ đáng kể so với các trình mô phỏng chức năng và thời gian/hiệu suất khác.

Ví dụ: *sim -fast* (trình mô phỏng chức năng nhanh nhất cho SimpleScalar ) có thể mô phỏng hướng dẫn nhanh hơn 25 lần so với mô hình mô phỏng cấp chu trình chi tiết của SimpleScalar . Mô hình hiệu suất cấp chu kỳ của SimpleScalar được gọi là *sim- outorder* , là mô hình định thời vi kiến trúc chi tiết. Nó triển khai một bộ xử lý siêu vô hướng không theo thứ tự hỗ trợ việc suy đoán. Hầu hết các tham số thiết kế đều có thể được cấu hình bởi người dùng, ví dụ: số lượng và độ trễ của các đơn vị chức năng, hàng đợi lệnh và kích thước cửa sổ sắp xếp lại, độ trễ bộ nhớ, v.v. Một ví dụ khác về trình mô phỏng cấp chu trình là MSim [37], là một mô phỏng đa cấp. -môi trường mô phỏng vi kiến trúc có luồng dành cho bộ xử lý Alpha mô phỏng các thành phần đường ống chính. MSim dựa trên SimpleScalar .

***Trình mô phỏng hướng sự kiện:*** Trình mô phỏng hướng sự kiện mô phỏng mục tiêu dựa trên các sự kiện thay vì chu kỳ. Thông thường, họ sử dụng hàng đợi sự kiện. Mô phỏng chuyển sang thời điểm một sự kiện được lên lịch, dựa trên hàng đợi sự kiện, thay vì trải qua tất cả các chu kỳ. Bằng cách đó, trình mô phỏng có thể tiết kiệm thời gian bằng cách không mô phỏng các chu kỳ không có sự kiện theo lịch trình [38]. Thông thường, một số phần của trình mô phỏng được mô hình hóa ở cấp độ chu kỳ, trong khi những phần khác được điều khiển theo sự kiện. Ví dụ, công việc được thực hiện bởi Reilly và Edmondson [39] để mô phỏng hiệu suất của bộ vi xử lý Alpha. Một điểm quan trọng cần lưu ý ở đây là tài liệu thường không phân biệt giữa trình mô phỏng theo cấp độ chu trình và trình mô phỏng theo hướng sự kiện.

SESC [40], một trình mô phỏng tương đối nhanh là một ví dụ về trình mô phỏng thời gian hướng sự kiện hỗ trợ MIPS ISA. SESC hỗ trợ nhiều mô hình mô phỏng khác nhau như bộ xử lý đơn, bộ đa xử lý chip (CMP), bộ xử lý trong bộ nhớ (PIM). RSim [42] là một ví dụ khác về trình mô phỏng hiệu suất/thời gian theo hướng sự kiện . Nó được phát triển vào những năm 1990 và tập trung vào cả song song cấp độ hướng dẫn (ILP) và bộ đa xử lý bộ nhớ dùng chung. Độ chính xác chi tiết trong RSim đạt được với cái giá phải trả là tốc độ chậm [42]. Lấy mẫu mô phỏng vi kiến trúc (SMARTS) [20] và Flexus (Simics) [43] tạo thành cơ sở của trình mô phỏng thời gian cấp chu kỳ SimFlex [44]. Một số thành phần của trình mô phỏng này được điều khiển theo sự kiện nội bộ. SimFlex có thể thực hiện mô phỏng nhanh các hệ thống đơn xử lý và đa xử lý. Nó hỗ trợ nhiều mô hình bộ nhớ khác nhau, nhưng thực hiện mô hình CPU theo thứ tự đơn giản.

***Trình mô phỏng theo khoảng thời gian:*** Với sự chuyển hướng tập trung nghiên cứu sang các hệ thống đa lõi và nhiều lõi, các nhà nghiên cứu đã tìm kiếm các kỹ thuật mô phỏng mới giúp cân bằng độ chính xác và tốc độ mô phỏng làm lựa chọn thay thế cho các trình mô phỏng chỉ theo cấp độ chu kỳ và theo sự kiện. Ví dụ, mô phỏng khoảng [45] là một trong những kỹ thuật được đề xuất gần đây. Kỹ thuật này sử dụng thực tế là luồng lệnh thông thường qua đường ống có thể được chia thành các tập hợp khoảng thời gian dựa trên các sự kiện bị bỏ sót (lỗi bộ nhớ đệm, dự đoán sai nhánh). Các phần có mục đích đặc biệt của trình mô phỏng kiến trúc, như bộ dự đoán nhánh và hệ thống bộ nhớ, có thể được sử dụng để mô phỏng các sự kiện bị bỏ sót và tìm thời gian chính xác của chúng. Sau đó, những khoảng thời gian này cùng với mô hình phân tích được sử dụng để ước tính thời lượng cho mỗi khoảng thời gian hướng dẫn.

3) TÍCH HỢP THỜI GIAN VÀ CHỨC NĂNG MÔ PHỎNG

Các bộ mô phỏng chức năng thường được tích hợp với các bộ mô phỏng thời gian để đạt được mô hình mô phỏng linh hoạt và chính xác hơn. Hai loại trình mô phỏng có thể được kết hợp hoặc không với nhau. Kỹ thuật ghép nối các trình mô phỏng, trong đó các lệnh thực thi ở giai đoạn thực thi của đường ống được mô hình hóa, được gọi là *thực thi trong thực thi* . Điều này làm cho *việc thực thi đang thực thi trở thành* một kỹ thuật tương đối phức tạp so với việc tách rời. Mặt khác, nó có thể tăng độ chính xác của các hướng dẫn phụ thuộc vào thời gian được mô hình hóa như hoạt động đồng bộ hóa và IO [46]. gem5 [2] là một ví dụ về trình mô phỏng sử dụng kỹ thuật này. Để đơn giản hóa quá trình phát triển và giảm độ phức tạp của nó, các trình mô phỏng thường tách rời mô phỏng chức năng và thời gian (hiệu suất). Thông thường, một số phần mềm của bên thứ ba được sử dụng để mô phỏng chức năng. Ví dụ: Simics [28] được cả SimFlex [44] và GEMS [38] sử dụng để mô phỏng chức năng và Pin [34] được sử dụng bởi cả Graphite [47] và Sniper [6] (dựa trên Graphite) để mô phỏng chức năng . Để tách rời, có ba loại thực tế [33]: định hướng thời gian, chức năng đầu tiên và định thời gian đầu tiên. Trong trường hợp mô phỏng định hướng theo thời gian, mô hình định thời sẽ dẫn đầu mô phỏng và đưa ra chỉ dẫn cho mô hình chức năng để thực hiện các hướng dẫn. Điều này giúp cho các trình mô phỏng định hướng thời gian có thể mô hình hóa các đường dẫn suy đoán. Trình mô phỏng chức năng đầu tiên sử dụng các mô hình chức năng để tạo ra các dấu vết hướng dẫn được đưa vào mô hình định thời để rút ra mô phỏng chi tiết. Vì các mô hình chức năng chỉ thực hiện các hướng dẫn trên đường dẫn chính xác nên rất khó để mô hình hóa các đường dẫn suy đoán bằng loại mô phỏng này. Trong trường hợp trình mô phỏng định thời gian đầu tiên, trình mô phỏng định thời cũng thực thi các lệnh và sử dụng một mô hình chức năng riêng biệt để xác minh việc thực thi của nó. Hình 3(a), 3(b) và 3(c) minh họa ba loại này.

***Trình mô phỏng theo thời gian:*** Trong danh mục này, trình mô phỏng chức năng ghi lại trạng thái kiến trúc (ví dụ: giá trị thanh ghi và bộ nhớ) của bộ xử lý được mô phỏng. Trình mô phỏng thời gian, không có ý tưởng về các giá trị dữ liệu, lấy và sử dụng các giá trị này từ trình mô phỏng chức năng để thực hiện một nhiệm vụ cụ thể khi được yêu cầu [33]. Mô hình chức năng và mô hình định thời tương tác rất nhiều trong loại mô phỏng này vì mô hình định thời chỉ đạo mô hình chức năng và mô hình chức năng cung cấp các giá trị cho mô hình định thời. Sự tương tác này làm cho mô hình mô phỏng này phù hợp để mô hình hóa các kiến trúc có hành vi chức năng thay đổi linh hoạt, chẳng hạn như kiến trúc đa lõi [48]. Ví dụ: đối với lệnh tải, mô hình chức năng sẽ tính toán địa chỉ hiệu dụng của lệnh và mô hình định thời sử dụng địa chỉ này để xác định xem tải có gây ra lỗi bộ đệm hay không. Giá trị được trả về từ bộ đệm hoặc bộ nhớ cuối cùng sẽ được trình mô phỏng chức năng đọc. Asim [49] là một ví dụ về loại mô phỏng này.

***Trình mô phỏng chức năng đầu tiên:*** Trong mô hình mô phỏng này, trình mô phỏng chức năng chạy trước trình mô phỏng thời gian và tạo ra dấu vết lệnh (một luồng hướng dẫn) cung cấp cho trình mô phỏng thời gian trong thời gian chạy. Trong trường hợp các nhánh có điều kiện, bộ mô phỏng chức năng luôn đi theo đường dẫn chính xác và nó không thể mô phỏng hành vi của các bộ dự đoán nhánh [50]. Nếu có một nhánh bị dự đoán sai trong quy trình của trình mô phỏng thời gian, thì trình mô phỏng chức năng sẽ khôi phục trạng thái trước đó của nó trước nhánh đó và tiếp tục dọc theo đường dẫn bị dự đoán sai . Sau đó, đường ống phải được xả lại do nhánh dự đoán sai này. Vì trình mô phỏng thời gian luôn chậm hơn trình mô phỏng chức năng nên có thể có vấn đề về thứ tự trong khi mô phỏng nhiều hơn một luồng [50]. Ví dụ: thời gian mà mô hình chức năng đọc một giá trị bộ nhớ trong trường hợp lệnh *tải* có thể khác với thời điểm mô hình thời gian yêu cầu cùng một giá trị và điều này có thể dẫn đến việc đọc các giá trị khác nhau. Vấn đề này có thể được giải quyết bằng mô phỏng chức năng đầu cơ [33]. Trong kỹ thuật này, bất cứ khi nào mô hình định thời phát hiện dữ liệu mà nó đọc khác với dữ liệu mà mô hình chức năng đã đọc, nó sẽ yêu cầu mô hình chức năng khôi phục trạng thái của bộ xử lý về trạng thái trước lệnh tải và sau đó nó thực thi lệnh tải. với dữ liệu chính xác. Khi các mô hình chức năng và thời gian chạy song song, sẽ có cơ hội khai thác tính song song này để trình mô phỏng hoạt động tốt hơn. Loại mô phỏng này có hiệu suất tốt hơn nhiều so với các mô phỏng định hướng theo thời gian, bởi vì mô hình định thời không bắt buộc phải điều khiển mô hình chức năng ở mọi lệnh hoặc chu kỳ như trong các trình mô phỏng định hướng theo thời gian. SimWattch [51] là một ví dụ về các trình mô phỏng chức năng đầu tiên. SimWattch tích hợp Simics với Wattch [52]. Wattch dựa trên SimpleScalar và mô phỏng cả sức mạnh lẫn hiệu suất.

***Trình mô phỏng thời gian đầu tiên:*** Trong phương pháp này, trình mô phỏng thời gian chạy trước các trình mô phỏng chức năng [50]. Bộ mô phỏng thời gian mô phỏng vi kiến trúc của bộ xử lý đích ở cấp độ chu kỳ. Bộ mô phỏng thời gian thường sử dụng bộ mô phỏng chức năng để xác minh việc thực hiện chức năng của tất cả các lệnh. Lệnh sẽ bị loại bỏ trong trường hợp có sự trùng khớp giữa trạng thái kiến trúc của cả bộ mô phỏng chức năng và bộ mô phỏng thời gian. Trong trường hợp không khớp, trình mô phỏng thời gian sẽ phục hồi bằng cách xóa đường ống và khởi động lại quá trình tìm nạp lệnh theo lệnh có vấn đề. Như vậy, trình mô phỏng thời gian sẽ tiến triển về phía trước. Nếu những sự phục hồi này xảy ra thường xuyên, chúng có thể ảnh hưởng đến thời gian của hệ thống mô phỏng và do đó ảnh hưởng đến độ chính xác, tùy thuộc vào độ sâu của đường ống mô phỏng [50]. GEMS [38], Fe *S* 2 [53] và Multi2sim [3] là một số ví dụ về tính toán thời gian của các trình mô phỏng đầu tiên.

*B. PHÂN LOẠI MÔ PHỎNG DỰA TRÊN*

*PHẠM VI MỤC TIÊU*

Một yếu tố khác cần xem xét khi phân loại trình mô phỏng là phạm vi của hệ thống đích đang được mô phỏng. Trình mô phỏng có thể được phân thành hai loại dựa trên phạm vi:

1) MÔ PHỎNG HỆ THỐNG ĐẦY ĐỦ

Với bất kỳ tệp nhị phân nào của hệ điều hành (OS) được hỗ trợ, trình mô phỏng hệ thống đầy đủ có thể khởi động hoàn toàn hệ điều hành đó và chạy các điểm chuẩn ứng dụng trên hệ điều hành đó giống như chúng chạy bình thường trên máy mục tiêu thực. Hình 4(a) hiển thị chức năng cơ bản của trình mô phỏng hệ thống đầy đủ. Trình mô phỏng toàn hệ thống mô phỏng tất cả các thiết bị I/O, bộ nhớ và kết nối mạng cần thiết để khởi động và chạy bất kỳ hệ thống nào. Các ứng dụng chạy trên hệ thống đích mô phỏng thực thi lệnh gọi hệ thống của chúng trực tiếp trên hệ thống đích [26]. Kết quả là hình thức mô phỏng này phức tạp và tốn thời gian. Lý tưởng nhất là không nên sửa đổi hệ điều hành , nhưng đôi khi các tệp nhị phân của hệ điều hành được tùy chỉnh để làm cho quá trình khởi động hệ điều hành ít tiêu tốn tài nguyên hơn.

Trình mô phỏng toàn hệ thống có thể là trình mô phỏng thời gian hoặc không. Ví dụ: gem5 [2] là một ví dụ về trình mô phỏng thời gian toàn hệ thống , trong khi SimOS [54] là trình mô phỏng chức năng toàn hệ thống. gem5 có khả năng khởi động các hệ điều hành Linux, Solaris và Android trên phần cứng cụ thể mà nó hỗ trợ. SimOS được phát triển vào cuối những năm 90, là một trong những trình mô phỏng toàn hệ thống đầu tiên (hiện không hoạt động). Bộ Sunflower [55] bao gồm một bộ mô phỏng toàn hệ thống với vi kiến trúc và mô hình IO. Simics có thể khởi động hệ điều hành chưa sửa đổi. SimFlex [44], ML- RSim [56], MARSSx86 [4] và PTLsim [5] là những ví dụ khác về loại mô phỏng này.

2) MỨC ĐỘ ỨNG DỤNG/CHẾ ĐỘ NGƯỜI DÙNG MÔ PHỎNG

Những trình mô phỏng này chỉ chạy các ứng dụng đích thay vì mô phỏng một hệ điều hành hoàn chỉnh. Chúng mô phỏng bộ vi xử lý và các thiết bị ngoại vi hạn chế. Trong loại trình mô phỏng này, các lệnh gọi hệ thống thường bị trình mô phỏng bỏ qua và được hệ điều hành máy chủ cơ bản phục vụ, như trong Hình 4(b). Có thể không có vấn đề gì khi chỉ mô phỏng mã mã người dùng cho các điểm chuẩn thực thi mã cấp hệ thống trong một khoảng thời gian ngắn (ví dụ: tính toán các điểm chuẩn chuyên sâu như SPEC CPU2006 và CPU2017) [33]. Tuy nhiên, đối với các điểm chuẩn dành nhiều thời gian để thực thi mã chế độ hệ thống, mô phỏng chế độ người dùng là không đủ– (ví dụ: các điểm chuẩn liên quan đến máy chủ như Web-Bench và NetBench , các điểm chuẩn hiệu suất trong thế giới thực chẳng hạn như SYSMARK và các điểm chuẩn cơ sở dữ liệu và xử lý giao dịch như TPC-C). Đối với khối lượng công việc đa luồng, cần tính đến việc lập kế hoạch cho hệ điều hành vì nó ảnh hưởng đến hiệu suất của khối lượng công việc. Vì vậy, cần phải mô phỏng các hiệu ứng ở cấp độ hệ điều hành để có được ước tính hiệu suất tốt hơn. Mặt khác, trình mô phỏng cấp ứng dụng thường ít phức tạp và nhanh hơn so với trình mô phỏng toàn hệ thống. SimpleScalar là ví dụ nổi tiếng nhất về trình mô phỏng cấp ứng dụng. SESC [40], Sniper [6] và RSim [42] là những ví dụ khác về trình mô phỏng chỉ dành cho ứng dụng.

*C. PHÂN LOẠI MÔ PHỎNG DỰA TRÊN ĐẦU VÀO*

*ĐẾN MÁY MÔ PHỎNG*

Chúng ta có thể phân loại trình mô phỏng thành hai loại dựa trên đầu vào của chính trình mô phỏng, dấu vết hoặc tệp thực thi.

1) Bộ mô phỏng điều khiển theo dấu vết

Các tệp theo dõi được sử dụng làm đầu vào cho trình mô phỏng điều khiển theo dấu vết. Các tệp theo dõi này là các luồng hướng dẫn được ghi trước, được thực thi theo điểm chuẩn với một số đầu vào cố định. Khi điểm chuẩn thực thi trên số liệu thống kê của máy thực, bao gồm mã lệnh, địa chỉ dữ liệu, địa chỉ đích nhánh, v.v. đều được ghi lại trong tệp theo dõi. Mô hình điều khiển theo dấu vết làm cho việc triển khai trình mô phỏng trở nên đơn giản. Trình mô phỏng theo hướng dấu vết có thể được gỡ lỗi dễ dàng vì kết quả thử nghiệm có thể được sao chép. Kích thước của tệp dấu vết có thể rất lớn, điều này đặt ra giới hạn về tổng số lệnh trong mỗi tệp dấu vết và/hoặc số lượng tệp dấu vết được sử dụng cùng một lúc và có thể dẫn đến thời gian mô phỏng chậm hơn [33], [57]. Các kỹ thuật lấy mẫu dấu vết và giảm dấu vết khác nhau [58] được sử dụng để giải quyết vấn đề về kích thước lớn của tệp dấu vết. Ngoài ra, những trình mô phỏng này thường không lập mô hình thực hiện các hoạt động suy đoán sai. mã, có thể ảnh hưởng đến kết quả ước tính hiệu suất của các cấu trúc như bộ dự đoán nhánh. Để giải quyết vấn đề dự đoán sai nhánh, các kỹ thuật như xây dựng lại đường đi bị dự đoán sai [59] được sử dụng.

Các mô hình theo dõi không bao gồm những thay đổi trong thời gian chạy về hành vi của các ứng dụng đa luồng [60]. Điều này trở thành một vấn đề rõ ràng hơn nếu mô phỏng theo hướng dấu vết được chạy cho một hệ thống đa bộ xử lý mô phỏng khác với hệ thống được sử dụng để thu thập dấu vết. Nên tránh mô phỏng theo hướng theo dõi đối với các hệ thống song song và phụ thuộc vào thời gian như Goldschmidt et al đã nhấn mạnh. [61].

Shade [62] là một trình mô phỏng tập lệnh hướng dẫn theo dõi, hỗ trợ các hệ thống SPARC và MIPS. Bóng râm cũng được sử dụng để tạo dấu vết. Simplescalar cũng có khả năng chạy mô phỏng từ các tệp dấu vết. Cheetah [63] là một trình mô phỏng theo hướng mô phỏng các cấu hình bộ đệm khác nhau. MASE [64] là một ví dụ khác của loại mô phỏng này. Rất khó để các trình mô phỏng điều khiển theo dấu vết mô hình hóa các thay đổi trong thời gian chạy trong hoạt động của các ứng dụng đa luồng [60], [61]. Tuy nhiên, gần đây, một số công trình nghiên cứu đã được đưa ra để sử dụng hiệu quả các trình mô phỏng theo hướng cho khối lượng công việc đa luồng, [65], [66].

2) MÔ PHỎNG ĐIỀU HÀNH THỰC HIỆN

Trình mô phỏng hướng thực thi không sử dụng tệp theo dõi. Thay vào đó, những trình mô phỏng này sử dụng trực tiếp các tệp nhị phân hoặc tệp thực thi của điểm chuẩn cho các máy mục tiêu được mô phỏng. Những trình mô phỏng này có thể mô phỏng các hướng dẫn bị suy đoán sai không giống như các trình mô phỏng điều khiển theo dấu vết. Tuy nhiên, chúng phức tạp hơn so với các mô phỏng điều khiển theo dấu vết . SimpleScalar [27] thuộc loại mô phỏng này. Rsim [42], một trình mô phỏng hướng sự kiện riêng biệt dựa trên thư viện YACSIM [67], cũng diễn giải các tệp thực thi của ứng dụng thay vì các tệp theo dõi. SESC và ESESC [68] là những ví dụ khác về loại mô phỏng này.

Thông thường, người dùng quan tâm đến hiệu suất của các vùng mã được chọn thay vì toàn bộ điểm chuẩn. Kỹ thuật *thực thi trực tiếp/gốc* có thể giúp ích về mặt này. Trong thực thi trực tiếp, trình mô phỏng chỉ mô phỏng các phần mã cụ thể (hoặc *vùng quan tâm* ) của ứng dụng và thực thi phần còn lại của ứng dụng trực tiếp trên máy chủ [57]. Trong trường hợp này, cả hệ thống đích và máy chủ đều phải có cùng kiến trúc tập lệnh (ISA) để thực hiện việc thực thi nguyên gốc. Kỹ thuật này còn được gọi là *đồng mô phỏng* . PTLsim [5] sử dụng phương pháp này để tăng tốc độ mô phỏng. Tang [69], Proteus [70] và FAST [36] cũng sử dụng phương pháp này.

*D. CÁC LOẠI MÔ PHỎNG KHÁC*

Ngoài các cách phân loại trình mô phỏng đã nói ở trên, trình mô phỏng cũng có thể được phân loại dựa trên các khía cạnh khác hoặc chuyên môn của chúng như sau:

1) Bộ mô phỏng đa bộ xử lý/đa lõi

Gần đây, các hệ thống đa bộ xử lý/đa lõi đã trở nên phổ biến. Trình mô phỏng đa bộ xử lý phức tạp hơn trình mô phỏng bộ đơn bộ xử lý vì chúng phải đối mặt với những thách thức trong việc giữ cho các vùng quan trọng của ứng dụng nhất quán cho tất cả bộ xử lý/lõi và lập lịch trình của các quy trình [26]. Trình mô phỏng có thiết kế mô-đun có khả năng mô phỏng hệ thống đa bộ xử lý tốt hơn vì chúng có thể dễ dàng khởi tạo các bộ xử lý khác nhau và các mô-đun tương ứng để mô phỏng hệ thống đa lõi. Có hai cách tiếp cận chính để mô phỏng các mục tiêu song song: mô phỏng tuần tự và mô phỏng song song. Trong trường hợp mô phỏng tuần tự, chỉ có một luồng mô phỏng để mô phỏng tất cả các lõi mục tiêu. Trình mô phỏng trong trường hợp này mô phỏng các lõi theo kiểu vòng tròn [26]. Trong trường hợp phương pháp mô phỏng song song, các luồng mô phỏng khác nhau được sử dụng để mô phỏng các lõi khác nhau. Phương pháp này tăng tốc độ mô phỏng nhưng khó thực hiện do những thách thức chung của việc phát triển phần mềm đa luồng.

SimOS [54] và Simics [28] đều hỗ trợ mô phỏng đa bộ xử lý. SimCA [71] là một trình mô phỏng không còn được duy trì nhưng được phát triển dựa trên SimpleScalar mô hình không theo thứ tự. Nó tập trung vào kiến trúc bộ xử lý đa luồng. MINT [41] là gói phần mềm được thiết kế để xây dựng trình mô phỏng phân cấp bộ nhớ theo sự kiện cho bộ đa xử lý. Nó chỉ chạy trên các máy dựa trên MIPS và hỗ trợ mô phỏng MIPS. PTLsim hỗ trợ mô phỏng đa luồng và đa bộ xử lý. Augmint [72] là môi trường mô phỏng đa bộ xử lý điều khiển thực thi có sẵn công khai dành cho kiến trúc Intel x86. Tuy nhiên, MINT là nền tảng của Augmint ; Augmint cũng điều chỉnh kỹ thuật tăng cường [73] của Tango Lite. Trong phương pháp tăng cường này, ứng dụng được tăng cường bằng mã thiết bị tại thời điểm biên dịch. Mã thiết bị này cập nhật đồng hồ mô phỏng và tạo ra các sự kiện cho mô phỏng. ZSim [7], ESESC [68], SESC [40] và SimCore [29] là những ví dụ khác về trình mô phỏng hỗ trợ mô phỏng đa lõi.

2) MÁY MÔ PHỎNG NĂNG LƯỢNG VÀ ĐIỆN

Với nhu cầu cấp thiết là xây dựng các bộ xử lý và hệ thống máy tính tiết kiệm năng lượng, tầm quan trọng của các bộ mô phỏng năng lượng và công suất ngày càng tăng. Ngày nay có rất nhiều ví dụ về các mô phỏng như vậy. Wattch [52] được sử dụng rộng rãi để mô phỏng công suất tiêu thụ. Nó dựa trên Simplescalar và được thiết kế để kiểm tra và tối ưu hóa khả năng tiêu thụ điện năng cũng như mức tiêu thụ năng lượng của vi kiến trúc Alpha. Nó cũng có thể được hợp nhất với các trình mô phỏng khác. Ví dụ: SimWattch tích hợp Simics và Wattch . CACTI [74] là một ví dụ khác mô phỏng sức mạnh và diện tích cho các cấu trúc giống như bộ nhớ đệm. McPAT [17] có thể mô phỏng thời gian, diện tích và sức mạnh của bộ xử lý đa lõi. Một số ví dụ khác là Powertimer [75], PowerAnalyzer [76] và SimplePower [77]. SESCTherm [78] và Hotspot [79] là hai ví dụ khác của loại mô phỏng này mô hình hóa các hiệu ứng nhiệt ở cấp độ kiến trúc vi mô. Power Blurring [80] là một mô hình tính toán nhiệt độ khác, được phát triển dựa trên phương pháp tích chập ma trận, để giảm thời gian tính toán. Ziabari và cộng sự. [81] đã so sánh Power Blurring với HotSpot và SESCTherm . Các thí nghiệm của họ đã chỉ ra rằng kỹ thuật Power Blurring có thể đạt được độ chính xác cao hơn để tạo ra các cấu hình nhiệt độ trong khoảng thời gian ngắn hơn. ESESC [68] sử dụng McPAT và Hotspot đã được sửa đổi để mô phỏng năng lượng.

3) MÁY MÔ PHỎNG CHUYÊN DỤNG/TĂNG TỐC

Hiện có nhiều trình mô phỏng chuyên dụng có khả năng mô phỏng các phần kiến trúc/vi kiến trúc của bộ xử lý. Trong số đó, bộ mô phỏng bộ nhớ và mạng trên chip (NoC) là phổ biến nhất. Các trình mô phỏng chuyên dụng chỉ có khả năng thực hiện một số loại lệnh nhất định và thường sử dụng dấu vết của các lệnh cụ thể của các chương trình/điểm chuẩn đã thực thi làm đầu vào. Chúng dễ phát triển hơn và có thể đưa ra ý tưởng hay về hoạt động của các bộ phận cụ thể của bộ xử lý; tuy nhiên, chúng kém chính xác hơn vì chúng không mô phỏng toàn bộ bộ xử lý và sự tương tác với các bộ phận khác của bộ xử lý. Chúng thường không mô phỏng các hướng dẫn 'ngoài đường dẫn' trong trường hợp các nhánh được dự đoán sai , chẳng hạn như trong các trình mô phỏng dự đoán nhánh.

Trình mô phỏng bộ nhớ mô phỏng việc truy cập dữ liệu và lệnh vào bộ nhớ. Hầu hết các trình mô phỏng bộ nhớ hiện có đều có bản chất được điều khiển theo dấu vết, trong đó các tệp dấu vết chỉ chứa các luồng truy cập bộ nhớ. Ví dụ: DRAMSim là một trình mô phỏng thời gian có thể mô phỏng các loại bộ nhớ khác nhau như DDR, SDRAM, DRDRAM, v.v. [83]. DRAMSim cũng có thể được tích hợp với các trình mô phỏng khác. Cachesim5 [85] và Dinero IV [84] là những ví dụ về trình mô phỏng chỉ mô phỏng các truy cập bộ đệm. Mạng mô phỏng trên chip mô phỏng cơ sở hạ tầng truyền thông của bộ xử lý. Thiết kế bộ xử lý nhiều lõi mới nổi đòi hỏi mạng trên chip nhanh hơn/ít tắc nghẽn hơn. Vì vậy, trình mô phỏng NoC ngày càng được xây dựng và sử dụng nhiều hơn. Chúng có khả năng mô phỏng các loại mạng khác nhau trên chip, cấu trúc liên kết, chính sách định tuyến, v.v.

Trình mô phỏng máy gia tốc đã được sử dụng để mô phỏng hành vi của các chương trình hoặc các phần chương trình được tăng tốc bằng bộ xử lý đồ họa (GPU), mạch tích hợp dành riêng cho ứng dụng (ASIC), bộ xử lý tín hiệu số (DSP), mảng cổng lập trình trường (FPGA), xử lý dữ liệu gần và trong bộ nhớ , v.v. Bộ tăng tốc gần đây đã được tích hợp với các bộ xử lý trên cùng một chip hoặc trên chip hệ thống (SoC). Ngoài ra, chúng còn được đề xuất kết hợp chặt chẽ với bộ xử lý. Việc mô phỏng các bộ tăng tốc cũng như bộ xử lý mang lại cái nhìn đầy đủ về hiệu suất của các điểm chuẩn. Bảng 1 cho thấy các loại mô phỏng máy gia tốc và chuyên dụng khác nhau hiện có cùng với một số ví dụ. Sự tương tác giữa bộ tăng tốc và bộ xử lý cũng rất quan trọng để mô phỏng, một số trình mô phỏng kiến trúc máy tính bao gồm (hoặc có thể được tích hợp vào) bộ mô phỏng bộ tăng tốc như gem5 [2] và Multi2Sim [3].

4) MÔ PHỎNG MÔ-ĐUN

Trình mô phỏng mô-đun, thay vì có thiết kế nguyên khối, lại chứa các mô-đun độc lập cho các phần khác nhau của bộ xử lý có thể được khởi tạo và liên kết với các khối khác của hệ thống mô phỏng. Những trình mô phỏng này có thể được gỡ lỗi dễ dàng hơn và phù hợp hơn với các thiết kế phức tạp so với các trình mô phỏng không mô-đun. Tính mô-đun này làm cho trình mô phỏng dễ quản lý hơn. Môi trường mô phỏng Liberty (LSE) [107] là một ví dụ về mô phỏng mô-đun. LSE sử dụng một chức năng phần mềm duy nhất cho từng thành phần phần cứng và người thiết kế có thể sử dụng các thành phần đó được kết nối theo cấu trúc phân cấp để xây dựng bất kỳ hệ thống phức tạp nào. Asim [49] là một ví dụ khác về loại mô phỏng này. Nó là một trình mô phỏng chế độ người dùng mở rộng SimpleScalar thành các thành phần mô-đun bên trong chính trình mô phỏng đó. MicroLib [21], M5 [46], Soonergy [108], [109] và gem5 là một số ví dụ khác về trình mô phỏng mô-đun, cung cấp khả năng sử dụng lại các mô-đun của một thành phần bộ xử lý nhất định trong hệ thống máy tính mới.

**III. ĐÁNH GIÁ MÁY MÔ PHỎNG**

Việc đánh giá các trình mô phỏng là một thách thức vì cần xem xét các số liệu mâu thuẫn nhau. Eeckhout [33] thể hiện sự đánh đổi mô phỏng như một viên kim cương với các yếu tố trái ngược nhau: độ chính xác, thời gian đánh giá, thời gian phát triển và phạm vi bao phủ. Trong bài viết này, chúng tôi xem xét sáu số liệu đánh đổi có thể được sử dụng để đánh giá trình mô phỏng: độ chính xác, hiệu suất, mức độ chi tiết, tính dễ phát triển, tính linh hoạt và thân thiện với người dùng.

Độ chính xác của trình mô phỏng đề cập đến độ chính xác về hiệu suất của mục tiêu được mô phỏng so với phần cứng thực. Hiệu suất của trình mô phỏng đề cập đến việc trình mô phỏng có thể chạy nhanh hay chậm trong khi mô phỏng kiến trúc đích. Mức độ chi tiết thể hiện số lượng và mức độ chi tiết mà trình mô phỏng bao gồm trong khi thể hiện kiến trúc đích. Không dễ để đạt được kết quả tốt nhất trong tất cả những sự đánh đổi này vì hầu hết chúng đều có những số liệu trái ngược nhau. Tính linh hoạt đề cập đến cả khả năng cấu hình của trình mô phỏng và mức độ linh hoạt của trình mô phỏng trong việc sửa đổi (hoặc thêm mới) cấu trúc. Sự thân thiện với người dùng đề cập đến việc người dùng dễ dàng học cách sử dụng trình mô phỏng, sửa đổi nó và chạy các thử nghiệm khác nhau như thế nào. Bảng 2 so sánh bốn sự cân bằng đầu tiên cho các loại mô hình mô phỏng khác nhau được mô tả trước đó trong phần II . Tính linh hoạt và thân thiện với người dùng bị ảnh hưởng nhiều hơn bởi việc triển khai mô phỏng so với danh mục trình mô phỏng và do đó bị loại khỏi so sánh.

**IV. TỔNG HỢP CÁC BỘ PHƯƠNG PHÁP XỬ LÝ HIỆN CÓ**

Tồn tại một số mô phỏng kiến trúc bộ xử lý. Bảng 3 dưới đây tóm tắt các mô phỏng kiến trúc/vi kiến trúc máy tính hiện có; tất cả các trình mô phỏng đều là nguồn mở ngoại trừ Simics. Nó tóm tắt các khía cạnh khác nhau của trình mô phỏng bao gồm các máy chủ và mục tiêu được hỗ trợ, v.v. Ngoài ra, nó mô tả đặc điểm của trình mô phỏng dựa trên phân loại phân loại được mô tả trong phần II ở trên.

**V. THÁCH THỨC MÔ PHỎNG**

Những thách thức chính trong mô phỏng liên quan đến hiệu suất và độ chính xác của trình mô phỏng [121]. Phần này mô tả những thách thức này và đề xuất các chiến lược để giải quyết chúng.

*A. MÔ PHỎNG CHẬM*

Kiến trúc sư máy tính và nhà thiết kế hệ thống dựa vào mô phỏng với thời gian chính xác để đưa ra quyết định thiết kế phù hợp. Việc mô phỏng một ứng dụng chỉ có thể mất nhiều thời gian – từ vài giờ đến vài ngày. Lý do chính khiến thời gian mô phỏng dài là do sự phức tạp của các vi kiến trúc hiện đại được mô phỏng và độ dài của các chương trình ngày nay, bao gồm hàng tỷ nghìn tỷ lệnh. Với sự ra đời của các hệ thống đa bộ xử lý và đa lõi, các trình mô phỏng phải theo dõi các tài nguyên được chia sẻ và xử lý việc đồng bộ hóa vốn tiêu tốn nhiều tài nguyên. Điểm chuẩn cũng trở nên phức tạp hơn so với trước đây [33]. Ví dụ: điểm chuẩn CPU SPEC ngày càng trở nên phức tạp hơn. Số lượng lệnh động trên mỗi điểm chuẩn trung bình là 2,5 tỷ lệnh trong CPU89, nó tăng lên 230 tỷ lệnh trong CPU2000 [122] và lên 2,5 nghìn tỷ lệnh trong CPU2006 [123]. CPU2017 có số lệnh động trung bình cao hơn 10 lần so với CPU2006 [124]. Ngày nay, các ứng dụng ngày càng trở nên đa luồng để sử dụng bộ xử lý đa lõi một cách hiệu quả và việc mô phỏng bộ xử lý đa lõi sẽ tiêu tốn nhiều tài nguyên và thời gian hơn so với bộ xử lý lõi đơn. Nhiều kỹ thuật và chiến lược đổi mới đã được đề xuất để tăng tốc độ mô phỏng. Một số trong số họ được thảo luận dưới đây:

1) MÔ PHỎNG MẪU

Một trong những kỹ thuật được sử dụng nhiều nhất để tăng tốc mô phỏng là Lấy mẫu. Trong mô phỏng lấy mẫu, thay vì mô phỏng toàn bộ điểm chuẩn, chỉ một số lượng nhỏ mẫu được mô phỏng. Các mẫu này là các nhóm hướng dẫn, được coi là đại diện cho toàn bộ điểm chuẩn. Việc lựa chọn điểm lấy mẫu có thể được thực hiện theo hai cách: (1) lấy mẫu thống kê và (2) lấy mẫu có mục tiêu. Một cách tiếp cận để lấy mẫu thống kê là chọn ngẫu nhiên các mẫu từ toàn bộ luồng lệnh để thu được các mẫu không thiên vị. Phương pháp lấy mẫu thống kê khác là lấy mẫu định kỳ, chọn các đơn vị lấy mẫu theo các khoảng thời gian đều đặn trong toàn bộ chương trình. Ví dụ: lấy mẫu định kỳ được sử dụng bởi SMARTS [20] và Flexus [43].

Lấy mẫu có mục tiêu chọn các điểm lấy mẫu sau khi phân tích hành vi của chương trình. Các điểm/đơn vị lấy mẫu đơn được chọn từ mỗi giai đoạn (một giai đoạn là một nhóm gồm một số lượng lớn các lệnh liên tiếp có hành vi tương tự). Trọng lượng của các pha thường được tính toán và xem xét khi chọn điểm lấy mẫu. Vì việc lấy mẫu mục tiêu sử dụng hoạt động của chương trình để chọn các điểm lấy mẫu nên có thể việc lấy mẫu mục tiêu có thể dẫn đến số lượng tổng số mẫu ít hơn so với lấy mẫu thống kê. Tuy nhiên, việc lấy mẫu mục tiêu không thể cung cấp độ tin cậy cho ước tính hiệu suất [33]. SimPoint [125] là một công cụ tuân theo phương pháp lấy mẫu có chủ đích và sử dụng các khối cơ bản để phát hiện các giai đoạn của chương trình. SimPoint kết hợp các khối cơ bản thành các khoảng, sau đó sử dụng khoảng cách Manhattan để tìm ra sự tương đồng giữa các khoảng khác nhau nhằm xác định các giai đoạn của chương trình; mỗi pha chứa nhiều khoảng [126]. Các khoảng thời gian từ mỗi giai đoạn được chọn làm điểm lấy mẫu cho mô phỏng để thể hiện bức tranh hoàn chỉnh về việc thực hiện chương trình. Mỗi điểm lấy mẫu được gọi là điểm mô phỏng hoặc đôi khi là *SimPoint* . Yi *và cộng sự.* [127] đã thực hiện so sánh Simpoint và SMARTS. Nghiên cứu của họ chỉ ra rằng SMARTS chính xác hơn nhưng chậm hơn SimPoint .

Có hai thách thức liên quan đến kỹ thuật mô phỏng lấy mẫu [33], [128]. Thử thách đầu tiên là cung cấp chính xác điểm lấy mẫu với hình ảnh bắt đầu của trạng thái kiến trúc (ASSI). Trình mô phỏng chức năng yêu cầu ASSI, trạng thái kiến trúc của bộ xử lý (nội dung thanh ghi và bộ nhớ), cho mỗi điểm lấy mẫu để đạt được đầu ra chính xác. Ngoài ra, trình mô phỏng thời gian (hiệu suất) sử dụng các điểm kiểm tra ASSI để mô phỏng thời gian chính xác, tức là ASSI phải càng gần với trạng thái kiến trúc của chương trình ở đầu điểm mô phỏng càng tốt để đạt được kết quả mô phỏng chính xác. Nhiều trình mô phỏng có tính năng *chuyển tiếp nhanh* sử dụng trình mô phỏng chức năng nhanh để xây dựng ASSI [33], [128]. Trình mô phỏng thời gian sử dụng mô phỏng chi tiết cho toàn bộ điểm lấy mẫu nhưng chuyển sang mô phỏng chức năng ở cuối mỗi điểm lấy mẫu cho đến khi đến điểm tiếp theo. Chuyển tiếp nhanh có thể tiêu tốn một lượng thời gian đáng kể khi các đơn vị lấy mẫu được đặt cách xa điểm bắt đầu của chương trình và cách xa nhau trong quá trình thực hiện lệnh động [33]. Ngoài ra, việc sử dụng chuyển tiếp nhanh và các khoảng thời gian mô phỏng chi tiết sẽ tuần tự hóa mô phỏng cho tất cả các điểm lấy mẫu. Điều này có nghĩa là, để xây dựng ASSI cho đơn vị lấy mẫu tiếp theo, người ta cần mô phỏng tất cả các đơn vị lấy mẫu trước đó và chuyển nhanh giữa các đơn vị lấy mẫu. Một kỹ thuật có thể được sử dụng để tăng tốc độ chuyển tiếp nhanh là *thực thi trực tiếp* như được triển khai trong PTLsim [5], trong đó chương trình được thực thi trực tiếp trên phần cứng gốc thay vì mô phỏng chức năng. *Điểm kiểm tra* [129] là một kỹ thuật khác có thể được sử dụng để giải quyết vấn đề khởi tạo trạng thái kiến trúc [33]. Điểm kiểm tra lưu trữ ASSI cho đến điểm lấy mẫu. Điểm kiểm tra này sau đó được tải từ đĩa trong quá trình mô phỏng được lấy mẫu. Điểm kiểm tra cũng cho phép mô phỏng song song thay vì chuyển tiếp nhanh [33]. Tuy nhiên, không gian cần thiết để lưu trữ các tệp điểm kiểm tra lớn trên đĩa là một nhược điểm liên quan đến điểm kiểm tra.

Để đạt được độ chính xác cao, mô phỏng được lấy mẫu cũng yêu cầu hình ảnh bắt đầu chính xác cho trạng thái vi kiến trúc chứa trạng thái của các bộ dự đoán nhánh, bộ nhớ đệm, v.v. Các chiến lược khác nhau để khởi động trạng thái bộ nhớ đệm được sử dụng [33], chẳng hạn như: khởi động liên tục, ước tính tỷ lệ lỗi bộ nhớ đệm , khởi động thích ứng tự giám sát , độ trễ tái sử dụng đường ranh giới và điểm kiểm tra. Các cấu trúc lõi của bộ xử lý như khối chức năng, trạm dự trữ, bộ đệm sắp xếp lại, v.v. cũng cần được làm nóng để đạt độ chính xác cao nhất [33]. Đối với các đơn vị lấy mẫu lớn, đây không phải là vấn đề quan trọng vì lõi của bộ xử lý không lưu giữ lịch sử lâu dài của các sự kiện như các bộ dự đoán nhánh và bộ nhớ đệm. Tuy nhiên, đối với các điểm lấy mẫu ngắn, điều quan trọng là phải làm nóng chính xác các cấu trúc này.

Một trong những vấn đề liên quan đến kỹ thuật lấy mẫu là chúng không có khả năng hoạt động với các ứng dụng đa luồng. Các công cụ như Simpoint [125] và SMARTS [20] không hỗ trợ khối lượng công việc đa luồng. Tuy nhiên, gần đây một số nghiên cứu đã cố gắng giải quyết vấn đề này cho các lớp ứng dụng đa luồng cụ thể. Ví dụ: BarrierPoint [130] tận dụng các rào cản đồng bộ hóa trong các ứng dụng đa luồng để lấy mẫu một số lần lặp trong khối lượng công việc giữa hai rào cản.

2) MÔ PHỎNG THỐNG KÊ

Mô phỏng thống kê [131] kết hợp mô phỏng chi tiết và phân tích. Trước hết, một hồ sơ thống kê chứa các đặc điểm quan trọng của chương trình được tính toán bằng các công cụ dựa trên dấu vết đơn giản. Hồ sơ thống kê được sử dụng để tạo dấu vết hướng dẫn có thể được cung cấp cho trình mô phỏng điều khiển theo dõi. Những dấu vết tổng hợp này có kích thước rất nhỏ và quá trình mô phỏng có thể diễn ra nhanh chóng. Bởi vì mô phỏng thống kê có thể không đủ chính xác để kiến trúc sư đưa ra quyết định thiết kế, nó không thể thay thế mô phỏng chi tiết [132]. Tuy nhiên, sẽ rất hữu ích nếu nhận ra các vùng thú vị trong chương trình để phân tích bổ sung. Sử dụng mô phỏng thống kê, sai số trung bình trong giá trị IPC có thể dao động từ 10% (đối với lõi đơn giản) đến 15% (đối với lõi linh hoạt) [131]. Tuy nhiên, để khám phá không gian thiết kế, độ chính xác tương đối của mô hình hiệu suất (độ chính xác tương đối đề cập đến độ chính xác trong những thay đổi hiệu suất được quan sát thu được bằng cách thay đổi các tham số vi kiến trúc) có thể quan trọng hơn độ chính xác tuyệt đối của nó vì chúng ta thường chỉ quan tâm đến việc biết tác động hiệu suất của một mô hình hiệu suất. thay đổi trong một tham số cụ thể. Eeckhout và cộng sự. [131] đã thử nghiệm mô phỏng thống kê bằng cách sử dụng điểm chuẩn SPECint95. Kết quả của họ cho thấy sai số IPC tương đối là dưới 0,9% trong nhiều trường hợp khác nhau. Mô phỏng thống kê cũng đã được áp dụng để mô hình hóa năng lượng và đánh giá hệ thống, ví dụ như trong Wattch .

3) MÔ PHỎNG SONG SONG

Việc song song hóa các trình mô phỏng có thể giảm đáng kể thời gian mô phỏng cho mỗi lần chạy. Nếu điểm kiểm tra được sử dụng trong mô phỏng lấy mẫu, thì có thể áp dụng tính song song để mô phỏng nhiều điểm lấy mẫu cùng một lúc; điều này được gọi là *mô phỏng lấy mẫu song song* [33]. Thứ hai, để sử dụng các bộ xử lý đa lõi phổ biến, có vẻ hấp dẫn khi đưa ra các trình mô phỏng đa luồng và có thể sử dụng các bộ xử lý song song. Mã của trình mô phỏng có thể được chia thành các luồng khác nhau, trong đó mỗi luồng có thể được ánh xạ tới từng lõi đích trong trường hợp mô phỏng kiến trúc đa lõi. Một vấn đề với mô phỏng song song là việc cân bằng giữa tốc độ và độ chính xác [33]. Đối với trình mô phỏng cấp chu kỳ, các luồng sẽ cần phải đồng bộ hóa sau mỗi chu kỳ, điều này có thể là rào cản trong việc đạt được hiệu suất dự kiến bằng cách sử dụng tính song song. Là một giải pháp, điều kiện đồng bộ hóa theo từng chu kỳ có thể được nới lỏng thành nhiều chu kỳ như một sự cân bằng giữa độ chính xác và tốc độ mô phỏng [33], [133]. BigSim [134] sử dụng mô phỏng song song để mô phỏng các máy có số lượng bộ xử lý lớn. Sniper [6], Graphite [47], Barra [32] và ZSim [7] là những ví dụ khác về trình mô phỏng song song.

4) MÔ PHỎNG TĂNG TỐC FPGA

Để tăng tốc quá trình mô phỏng, các bộ phận của trình mô phỏng có thể được triển khai trên Mảng cổng lập trình trường (FPGA). Trình mô phỏng có thể tận dụng tính song song chi tiết có sẵn trên FPGA để đạt được tốc độ mô phỏng cao hơn so với các trình mô phỏng phần mềm thuần túy [36]. Tuy nhiên, thời gian phát triển của trình mô phỏng dựa trên FPGA có thể lớn so với trình mô phỏng phần mềm, vì trình mô phỏng tăng tốc FPGA phải được viết bằng ngôn ngữ mô tả phần cứng (HDL). Chúng cũng không thể tham số hóa được như các trình mô phỏng phần mềm. HAsim [111] là một trình mô phỏng điều khiển thực thi theo thời gian, thực hiện cả mô hình chức năng và mô hình định thời trên FPGA. Một ví dụ khác là NHANH CHÓNG [36]; nó dựa trên chiến lược mô phỏng chức năng đầu tiên. Trình mô phỏng chức năng của FAST được triển khai trong phần mềm, trong khi trình mô phỏng thời gian của nó là trình mô phỏng dựa trên phần cứng và chạy trên FPGA. Gần đây, đã có bộ chuyển đổi trình mô phỏng RISC-V Chisel-to-Verilog chuyển đổi trình mô phỏng được viết bằng ngôn ngữ xây dựng phần cứng mới được phát triển bởi UC Berkeley, Chisel, sang Verilog HDL [135]. Sau đó, mô phỏng bộ xử lý có thể được chạy trực tiếp trên FPGA. Viết mã Chisel có thể tốn ít thời gian hơn so với viết mã Verilog; tuy nhiên, nó đòi hỏi phải học một ngôn ngữ mới. Tương tự, Fabscalar là một trình mô phỏng x86, được viết bằng HDL/CCC, cho phép người dùng làm việc với cấp độ chuyển đăng ký (RTL) được tham số hóa tổng hợp để mô phỏng các thiết kế x86 bằng cách sử dụng FPGA [136]. Tuy nhiên, trình mô phỏng không có khả năng cấu hình cao vì nó chỉ cho phép người dùng lựa chọn trong số các cấu trúc và tham số được hỗ trợ khác nhau đã được triển khai trong HDL.

*B. ĐỘ CHÍNH XÁC Kém*

Độ trung thực của mô phỏng phải là mối quan tâm nghiêm túc, khi xem xét sự phụ thuộc của các quyết định thiết kế chính vào kết quả mô phỏng. Đầu tiên, các nhà phát triển trình mô phỏng phải đảm bảo rằng trình mô phỏng hoạt động chính xác nếu chúng đang mô phỏng chức năng của mục tiêu. Thứ hai, số liệu thống kê về hiệu suất phải chỉ ra hiệu suất thực tế của mục tiêu. Thật không may, trình mô phỏng không phải lúc nào cũng chính xác và có thể có nhiều lỗi khác nhau. Có thể có ba loại lỗi khác nhau trong trình mô phỏng [137]: Lỗi mô hình hóa, lỗi đặc tả và lỗi trừu tượng.

Lỗi lập mô hình xảy ra khi chức năng mong muốn không được triển khai hoặc lập mô hình đúng cách trong trình mô phỏng. Một ví dụ về lỗi mô hình hóa là khi các hướng dẫn được cấu hình để có độ trễ khác với mục tiêu được mô hình hóa. Một ví dụ khác có thể là đưa ra hướng dẫn cho các trạm đặt chỗ một cách không theo thứ tự. Các lỗi mô hình hóa có thể được giảm bớt bằng cách thiết kế và thử nghiệm cẩn thận các cấu trúc được mô hình hóa. Lỗi có thể được giảm thiểu hơn nữa bằng cách sử dụng các chiến lược thiết kế và nguyên tắc công nghệ phần mềm phù hợp.

Thứ hai, lỗi đặc tả là do thiếu kiến thức về chức năng chính xác của mục tiêu. Lỗi đặc tả chỉ có thể được giảm nếu tài liệu đặc tả của mục tiêu có thể truy cập được. Nếu không biết một số thông số kỹ thuật nhất định của phần cứng thực, việc viết điểm chuẩn vi mô có thể giúp ước tính một số thông số kỹ thuật. Ví dụ: người ta có thể ước tính quy mô của trạm đặt chỗ bằng cách viết và chạy điểm chuẩn vi mô cho các trường hợp khác nhau.

Thứ ba, lỗi trừu tượng xảy ra khi các nhà phát triển triển khai thiết kế của họ ở mức trừu tượng cao hơn để đánh đổi các chi tiết thiết kế để có tốc độ tốt hơn hoặc để đơn giản hóa việc triển khai trình mô phỏng của họ. Để giảm lỗi trừu tượng, các nhà phát triển thường cân bằng tốc độ; Người viết mô phỏng có thể giảm lỗi trừu tượng bằng cách đưa thêm chi tiết vào mô hình mô phỏng của họ. Các công nghệ mới ngày nay với phần cứng nhanh hơn cho phép giảm hơn nữa các lỗi trừu tượng. Cain và cộng sự. [138] thảo luận về một số nguồn lỗi trừu tượng ảnh hưởng đến độ chính xác của mô phỏng. Đầu tiên, họ kết luận rằng các hiệu ứng của hệ điều hành là quan trọng, do đó, việc mô phỏng toàn bộ hệ thống có thể giúp mô phỏng chính xác hơn và thể hiện hành vi thực sự của mục tiêu, đồng thời giảm các lỗi trừu tượng. Họ cũng nhận thấy rằng độ chính xác của trình mô phỏng có thể bị ảnh hưởng bằng cách mô phỏng hành vi I/O ngay cả đối với bộ đơn xử lý. Một ví dụ khác về lỗi trừu tượng là không mô phỏng các đường dẫn suy đoán không chính xác, điều này có thể làm giảm độ chính xác của trình mô phỏng. Tuy nhiên, đối với một số ứng dụng thương mại nhất định và điểm chuẩn số nguyên CPU SEPC, người ta đã chứng minh rằng việc mô phỏng các đường dẫn không chính xác này chỉ ảnh hưởng đến hiệu suất 2% [138].

**VI. XÁC NHẬN CÁC MÔ PHỎNG**

Xác thực trình mô phỏng đề cập đến quá trình xác thực rằng trình mô phỏng thể hiện chính xác phần cứng mục tiêu. Xác thực trình mô phỏng là quan trọng để đảm bảo rằng trình mô phỏng không bao gồm các lỗi về mô hình hóa, đặc tả hoặc trừu tượng. Việc xác thực trình mô phỏng thường kết hợp việc lập mô hình phần cứng thực và so sánh kết quả của trình mô phỏng với kết quả của phần cứng thực, sau đó tính toán lỗi thử nghiệm. Nếu sai số thực nghiệm cao thì trước tiên phải xác định loại và nguồn sai số. Sau đó, trình mô phỏng sẽ được sửa đổi để sửa những lỗi đó. Quá trình này có thể được lặp lại cho đến khi đạt được sai số thử nghiệm có thể chấp nhận được. Trình mô phỏng đã được xác thực mang lại sự tin cậy cho người dùng về kết quả của họ và rất quan trọng đối với khả năng tái tạo kết quả. Gibson *và cộng sự.* [139] và Black và Shen [137] khuyến nghị rằng các nghiên cứu mô phỏng nên được so sánh với nền tảng phần cứng tham chiếu hoặc trình mô phỏng đã được xác thực. Một nghiên cứu gần đây [140], hiệu chỉnh trình mô phỏng MARSSx86 cho một bộ xử lý không đồng nhất mục tiêu cụ thể, kết luận rằng trình mô phỏng không được xác thực/không được hiệu chỉnh có thể dẫn đến sự khác biệt đáng kể giữa kết quả mô phỏng và thống kê hiệu suất kiến trúc thực.

Mặc dù việc xác thực các trình mô phỏng là quan trọng trước khi dựa vào kết quả của chúng, một số nhà nghiên cứu cho rằng không phải lúc nào cũng có thể thực hiện được việc xác nhận nghiêm ngặt và các trình mô phỏng không được xác thực vẫn có thể cung cấp những hiểu biết sâu sắc về các quyết định thiết kế [42]. Hughes *và cộng sự.* [42] lập luận rằng việc xác nhận các trình mô phỏng có thể không thực tế cho việc nghiên cứu và nhiều trình mô phỏng chưa được xác nhận có thể tỏ ra hữu ích và có giá trị trong việc nghiên cứu hiện tượng kiến trúc và hiệu suất tương đối. Việc xác thực một trình mô phỏng có thể là một công việc tẻ nhạt do: (1) thiếu một số chi tiết nhất định về bộ xử lý hiện đại khiến việc triển khai các hệ thống đó một cách chính xác trong trình mô phỏng là gần như không thể; (2) việc triển khai một số chi tiết của bộ xử lý hiện đại, ngay cả khi đã biết, có thể tốn thời gian và có thể dẫn đến thời gian mô phỏng chậm hơn; (3) mô hình hóa hệ thống mục tiêu mới chỉ là ý tưởng nghiên cứu nên khó có thể kiểm chứng. Trong những trường hợp như vậy, việc xác thực trình mô phỏng trở nên không thể [33].

Chúng ta có thể tìm thấy nhiều nỗ lực xác nhận cho các trình mô phỏng khác nhau trong tài liệu [14], [19]–[24], [60], [68], [140]–[144]. Những nỗ lực xác nhận này khác nhau ở chiến lược họ sử dụng. Hầu hết, các kiến trúc sư máy tính so sánh kết quả của trình mô phỏng của họ với hành vi hiệu suất của điểm chuẩn trên máy thật đang được mô phỏng. Đôi khi, họ cũng dựa vào kết quả được công bố cho một phần cứng cụ thể thay vì chạy thử nghiệm trên hệ thống thực. Desikan *và cộng sự.* [19] đã cố gắng xác thực mô hình không theo thứ tự của SimpleScalar dựa trên mô hình bộ xử lý Alpha 21264. Sai số thử nghiệm trung bình trong mô phỏng microbenchmark đã giảm từ 19,5% đối với đơn hàng sim- outorder xuống còn 2% sau khi xác thực đơn hàng sim- outorder cho bộ xử lý Alpha ( *Sim-alpha* ). *Sim-alpha* [145] dựa trên trình mô phỏng SimpleScalar và sử dụng mã từ sim- outorder , nhưng hầu như tất cả mô hình mô phỏng thời gian đều được viết từ đầu. Để xác thực điểm chuẩn vĩ mô (điểm chuẩn được lấy từ bộ SPEC-CPU2000 [146]), thứ tự mô phỏng dẫn đến sai số thử nghiệm trung bình là 36,7% so với DS-10L (máy dựa trên bộ xử lý alpha) [19]. ESESC [68], là phần mở rộng của trình mô phỏng SESC [40], được xác thực dựa trên Samsung Chromebook (chứa ARM A15) bằng cách sử dụng điểm chuẩn CPU SPEC. Họ đã sử dụng tiện ích *hoàn hảo* [147] trên Chromebook có bật bộ đếm hiệu suất để thu thập số liệu thống kê trên hệ thống thực. Kết quả của họ cho thấy trung bình 21% lỗi IPC (hướng dẫn trên mỗi chu kỳ). Perez và cộng sự. [21] đã so sánh ước tính hiệu suất của các mô-đun khác nhau của môi trường mô phỏng MicroLib với SimpleScalar để xác thực chúng. Walker *và cộng sự.* [144] đã đề xuất một phương pháp mới để tìm ra nguồn gốc của sự thiếu chính xác trong trình mô phỏng và xác nhận chúng bằng cách sử dụng phân cụm, phân tích tương quan và hồi quy. Công việc trước đây của chúng tôi [148] đã so sánh lỗi thử nghiệm của một số trình mô phỏng kiến trúc máy tính với vi kiến trúc Core-i7 của Intel. Một công việc gần đây của Jo *et al.* [149] giới thiệu *DiagSim* để phát hiện các chi tiết ẩn trong ba trình mô phỏng (gem5, Multi2Sim, MARSSx86) có thể tác động đáng kể đến kết quả mô phỏng. Ngoài phần khảo sát về trình mô phỏng, bài viết này còn thảo luận thêm các ví dụ về nỗ lực xác nhận cho các trình mô phỏng gần đây trong Phần VII và *so sánh thêm* hiệu suất tuyệt đối và tương đối của chúng trong Phần IX .

**VII. SO SÁNH CÁC MÔ PHỎNG X86 GẦN ĐÂY**

Chúng tôi đã chọn sáu trình mô phỏng: gem5 [2], MARSSx86 [4], Multi2Sim [3], Sniper [6], PTLsim [5] và ZSim [7] để nghiên cứu toàn diện vì những lý do sau:

• Các trình mô phỏng này có các mô hình mô phỏng khác nhau, nhưng tất cả chúng đều thuộc loại mô phỏng thời gian.

• Đây là những trình mô phỏng hiện đại đang được phát triển tích cực, ngoại trừ PTLsim . PTLsim không được phát triển tích cực nhưng vẫn được sử dụng cho đến ngày nay.

• Tất cả các trình mô phỏng này đều hỗ trợ x86 và các kiến trúc chính khác. Họ cũng có khả năng thực hiện mô phỏng chi tiết trên các phần được chọn của bất kỳ điểm chuẩn nào.

*A. đá quý5*

gem5 [2] là một công cụ mô phỏng toàn hệ thống theo hướng sự kiện, được sử dụng rộng rãi trong cả giới học thuật và ngành công nghiệp. Mặc dù gem5 là một trình mô phỏng hướng sự kiện, nhưng nó có thể theo dõi các sự kiện theo từng chu kỳ, điều này làm cho độ chính xác của nó có thể so sánh với trình mô phỏng cấp chu kỳ. Nó hỗ trợ nhiều ISA: ARM, x86, MIPS, SPARC, ALPHA, Power và RISCV. Nó sử dụng các mẫu CPU từ M5 [46] và các mẫu hệ thống bộ nhớ từ GEMS [38]. Chủ yếu có bốn mẫu CPU trong gem5: ' AtomicSimple ', ' TimingSimple ', 'Minor' (theo thứ tự) và 'O3' (không theo thứ tự). Hai mô hình đầu tiên ( AtomicSimple và TimingSimple ) là các mô hình bộ xử lý chu trình đơn không có bất kỳ cấu trúc đường ống nào. AtomicSimple mô hình hóa thời gian truy cập bộ nhớ nhưng TimingSimple thì không. Minor và O3 là các mô hình đường ống 'thực thi'. Các mô hình này cho phép định cấu hình nhiều giai đoạn đường ống và chiều rộng, đơn vị chức năng cũng như các cấu trúc đường ống khác của chúng. Mô hình 'O3' hỗ trợ đa luồng đồng thời (SMT). Gần đây, mô hình CPU máy ảo dựa trên kernel (KVM-CPU) cũng được giới thiệu trong gem5, cho phép mã mô phỏng trong toàn hệ thống chạy trên phần cứng thực tế, do đó tăng tốc độ mô phỏng đáng kể. CPU này có thể được sử dụng để chuyển tiếp nhanh qua các phần không quan trọng của mã mô phỏng.

Gutierrez *và cộng sự.* [22] và Butko *và cộng sự.* [141] đã đánh giá độ chính xác của gem5 để mô hình hóa các bộ xử lý thực tế dựa trên ARM ISA (vi kiến trúc Cortex A15 và A9). Gutierrez *và cộng sự.* thử nghiệm của SPEC CPU2006 cho thấy độ chính xác trung bình là 13% [22]. Butko *và cộng sự.* [141] đã nghiên cứu độ chính xác của gem5 để mô phỏng mục tiêu nhúng đa lõi. Tanimoto *và cộng sự.* [150] cũng chỉ ra một số vấn đề với việc triển khai gem5 không theo thứ tự. Walker và cộng sự. gem5 được xác thực dựa trên hai vi kiến trúc ARM [144]. Độ không chính xác thay đổi từ 1,39% đến 17,94% dựa trên các thử nghiệm của họ. Akram và Sawalha đã tính toán lỗi thử nghiệm cho gem5 với x86 ISA và chỉ ra một số nguồn không chính xác [148]. Tuy nhiên, không có nỗ lực xác thực đầy đủ cho x86 ISA.

*B. MARSSx86*

MARSSx86 là trình mô phỏng toàn hệ thống x86 [4] được mô hình hóa ở cấp độ chu kỳ. Mô hình đường ống chi tiết của MARSSx86 dựa trên PTLsim [5]. Ngoài ra, nhiều tối ưu hóa khác nhau để có hiệu suất tốt hơn và tính linh hoạt đã được thêm vào. MARSSx86 sử dụng môi trường mô phỏng toàn hệ thống dựa trên QEMU [151] để thực hiện mô phỏng toàn hệ thống của các hệ điều hành chưa sửa đổi. Nó hỗ trợ cả mô hình đường ống không theo thứ tự và theo thứ tự (IO). MARSSx86 cho phép mô phỏng các cấu hình không đồng nhất. Nó cũng hỗ trợ mô phỏng các thiết bị đầu vào/đầu ra theo thời gian thực.

Asri *và cộng sự.* [140] đã hiệu chỉnh MARSSx86 để mô phỏng máy Intel Core i7, tập trung vào các ứng dụng điện toán hiệu năng cao. Nghiên cứu của họ đã bộc lộ một số vấn đề nhất định (ví dụ: số lượng *µ* -ops được đánh giá quá cao, khi giải mã hướng dẫn thành *µ* -ops) bằng trình mô phỏng. Trình mô phỏng MARSSx86 được hiệu chỉnh cuối cùng được hiển thị có sai số trung bình dưới 10% đối với điểm chuẩn SPEC và PARSEC.

*C. Multi2Sim*

Multi2Sim là trình mô phỏng chủ yếu nhắm vào GPU và mô phỏng kiến trúc CPU-GPU [3]. Nó hỗ trợ nhiều ISA chẳng hạn như x86, MIPS, ARM và AMD Evergreen ISA. Multi2Sim chủ yếu bao gồm ba khối mô phỏng khác nhau: công cụ mô phỏng chức năng, trình mô phỏng chi tiết và mô-đun hướng sự kiện. Trình mô phỏng chi tiết và mô-đun hướng sự kiện cùng nhau thực hiện mô phỏng thời gian. Nó hỗ trợ các lõi xử lý đa luồng hoặc đơn luồng với đường dẫn không theo thứ tự ( OoO ). Nó không mô hình hóa đường ống IO. Bộ nhớ và mạng kết nối có thể được cấu hình linh hoạt. Multi2Sim tuân theo triết lý thiết kế của Simplescalar [27] cho một số mô-đun của nó. Hơn nữa, nó là trình mô phỏng thời gian đầu tiên giống như GEMS [38].

Multi2Sim không hỗ trợ mô phỏng toàn bộ hệ điều hành, nhưng nó có thể sử dụng các luồng động để mô phỏng các chương trình song song.

Việc xác thực Multi2Sim cho GPU đã được thực hiện bởi Ubal *et al.* [142]. Họ đã sử dụng AMD Radeon 5870 làm mẫu GPU mục tiêu và các ứng dụng AMD OpenCL SDK [152] để đo điểm chuẩn. Kết quả đã xác minh tính đúng đắn của chức năng ngoài việc đo tỷ lệ phần trăm lỗi trung bình trong thời gian thực hiện (5% đến 30%). Theo hiểu biết tốt nhất của chúng tôi, không có nỗ lực xác thực nào đối với CPU x86 cho trình mô phỏng này.

*D.sim PTL*

PTLsim [5] là một trình mô phỏng cấp chu kỳ có khả năng mô phỏng hệ điều hành hoàn chỉnh bằng cách sử dụng bộ ảo hóa Xen [153]. Nó sử dụng kỹ thuật đồng mô phỏng hoặc thực hiện trực tiếp, đã được thảo luận trước đây. Nó có khả năng mô hình hóa lõi OoO siêu vô hướng . Nó không mô hình hóa đường ống IO chi tiết. Mô hình lõi mặc định của PTLsim ( OoO superscalar) dựa trên đặc điểm của các hệ thống thực khác nhau như bộ xử lý P4 và Core 2 của Intel và bộ xử lý K8 của AMD.

Yourst [5] đã đánh giá tính chính xác của PTLsim . Anh ấy đã sử dụng một chiếc máy thật có bộ xử lý AMD Athlon 64 2,2 GHz làm tài liệu tham khảo. Anh ấy đã sử dụng rsync [154], một ứng dụng máy chủ khách, làm điểm chuẩn kiểm tra. Kết quả cho thấy độ thiếu chính xác của PTLsim trong nhiều trường hợp nhỏ hơn 5%.

*E. BẮN TỈA*

Sniper [6] là một trình mô phỏng song song nhanh sử dụng phương pháp mô phỏng khoảng thời gian đã thảo luận trước đó [45]. Sniper dựa trên Graphite [47], hỗ trợ nhiều mô hình một IPC khác nhau. Sniper hỗ trợ cả mô phỏng đường ống OoO và IO. Carlson và cộng sự. [6] Sniper được xác thực bằng máy Intel Xeon X7460. Họ cho thấy mức độ thiếu chính xác trung bình dưới 25% đối với bộ điểm chuẩn SPLASH-2. Sau này, để nâng cao hơn nữa độ chính xác của tay bắn tỉa, Carlson *et al.* [23] đã triển khai mô hình dựa trên cửa sổ lệnh trong Sniper. Trình mô phỏng cải tiến cho thấy độ chính xác lõi đơn là 11,1% so với hệ thống mục tiêu dựa trên Nehalem của Intel. Ban đầu, Sniper chỉ hỗ trợ x86, tuy nhiên, gần đây hỗ trợ cho RISC-V ISA đã được thêm vào trình mô phỏng [155].

*F. ZSim*

ZSim [7] là trình mô phỏng định thời cấp ứng dụng song song cho kiến trúc x86-64. Ban đầu nó được viết cho mô hình ZCache [156], nhưng đã phát triển thành một trình mô phỏng hiệu quả hơn. Nó tập trung nhiều hơn vào việc mô phỏng hệ thống phân cấp bộ nhớ và nhiều hệ thống lõi không đồng nhất (ISA đơn). Nó hỗ trợ mô hình hóa cả đường ống OoO và IO. Việc sử dụng rộng rãi bản dịch nhị phân động cho phép nó đạt được tốc độ mô phỏng rất cao. Xác thực của ZSim [143] sử dụng lõi Intel Westmere cho thấy lỗi trung bình là 10%. Sai số tuyệt đối trung bình đối với khối lượng công việc đa luồng là 11,2%. Các điểm chuẩn vi mô khác nhau, điểm chuẩn đơn luồng (SPECCPU2006) và đa luồng (PARSEC, SPLASH2) đã được sử dụng cho nỗ lực xác thực. Nghiên cứu xác thực cho thấy rằng các nguồn lỗi chính trong trình mô phỏng là bộ đệm mục tiêu nhánh được lý tưởng hóa và không có khả năng lập mô hình bộ đệm dịch thuật (không chính xác trong mô hình giao diện người dùng của đường ống).

*G. SO SÁNH TÍNH NĂNG CỦA MÁY MÔ PHỎNG ĐƯỢC CHỌN*

Trong công việc trước đây của chúng tôi [148], chúng tôi đã so sánh các tính năng của một số trình mô phỏng x86 đã thảo luận ở trên. Trong bài viết này, chúng tôi bổ sung thêm một trình mô phỏng nữa vào so sánh chi tiết của chúng tôi, MARSSx86, như được hiển thị trong Bảng 4 . gem5 có thể chạy trên số lượng hệ điều hành (ví dụ Linux, MacOS X, Solaris, OpenBSD) và kiến trúc (ví dụ x86, x86-64, ARM, SPARC, Alpha và PPC) cao nhất so với các trình mô phỏng khác. Multi2Sim hỗ trợ máy Linux, MacOS X với x86. Sniper, PTLsim , MARSSx86 và ZSim chạy trên máy x86 dựa trên Linux.

Tất cả các trình mô phỏng này đều hỗ trợ chuyển tiếp nhanh và khởi động bộ đệm ngoại trừ PTLsim . Sniper không có khả năng tự tạo điểm kiểm tra mà sử dụng công cụ Pin [34] và Simpoint để tạo điểm kiểm tra. Tất cả các trình mô phỏng này có thể tạo dấu vết thực thi trong quá trình mô phỏng. Mô phỏng đường ống không theo thứ tự phức tạp ở chế độ chi tiết được hỗ trợ bởi tất cả các trình mô phỏng, nhưng đường ống IO không được hỗ trợ bởi tất cả các trình mô phỏng như Multi2Sim và PTLsim . gem5 tạo ra số liệu thống kê hiệu suất mô phỏng rất chi tiết (ví dụ: chu kỳ khối và nhàn rỗi của tất cả các giai đoạn đường ống, hướng dẫn bị nén ở các giai đoạn khác nhau do dự đoán sai nhánh và vi phạm thứ tự bộ nhớ). Multi2Sim, PTLsim và MARSSx86 cũng đưa ra số liệu thống kê chi tiết nhưng chi tiết ít hơn so với gem5.

Trong gem5, MARSSx86, PTLsim và ZSim , hình phạt do dự đoán sai nhánh có thể được thay đổi bằng cách thay đổi độ sâu đường ống. Mặt khác, việc thay đổi hình phạt do dự đoán sai nhánh trong Sniper và Multi2Sim có thể được thực hiện bằng cách chỉ định trực tiếp hình phạt do dự đoán sai và độ trễ hướng dẫn. Sniper và gem5 hỗ trợ thang đo tần số và điện áp động (DVFS) để nghiên cứu hiệu ứng thời gian chạy đối với hiệu quả sử dụng năng lượng. Số liệu thống kê từ tất cả các trình mô phỏng này có thể được sử dụng để tạo ra các mô hình công suất/năng lượng như McPAT [17]. Về mặt hỗ trợ mô phỏng đa lõi không đồng nhất (HMP), ZSim và Sniper hỗ trợ mô phỏng các hệ thống HMP chỉ với một ISA duy nhất, trong đó bộ xử lý HMP có thể có các tham số lõi khác nhau như mô hình thực thi, tần số, độ rộng gửi, kích thước cửa sổ, v.v. gem5 có hiện được tích hợp với GPUsim để mô hình hóa mô phỏng không đồng nhất CPU-GPU [157]. Hơn nữa, mã của gem5 có thể được thay đổi một chút để hỗ trợ mô phỏng HMP đa ISA. Multi2Sim tích hợp CPU và các kiến trúc GPU khác nhau có thể được sử dụng để mô phỏng CPU-GPU. Về mặt cộng đồng và diễn đàn hỗ trợ, Sniper và gem5 có các nhóm hỗ trợ có quy mô khá lớn. Multi2Sim, ZSim và MARSSx86 cũng có những diễn đàn như vậy; tuy nhiên, diễn đàn hỗ trợ PTLsim không còn được tiếp tục nữa.

**VIII. PHƯƠNG PHÁP VÀ THÍ NGHIỆM**

Phần này thảo luận chi tiết về phương pháp thử nghiệm được áp dụng để so sánh kết quả của sáu trình mô phỏng được chọn với kết quả của phần cứng thực và tìm ra các lỗi thử nghiệm.

*A. HỆ THỐNG MỤC TIÊU*

Hệ thống mục tiêu được sử dụng cho các thử nghiệm của chúng tôi dựa trên vi kiến trúc Haswell của Intel (lõi i7-4770). Mặc dù tất cả các thông số cấu hình không được Intel công bố cho hệ thống này nhưng chúng tôi đã phải dựa vào các nguồn khác để cấu hình các trình mô phỏng phù hợp với mục tiêu này. Chúng tôi đã sử dụng cả tài liệu Intel [158] và một số tài nguyên khác [159]–[161] để định cấu hình trình mô phỏng cho mô hình Haswell. Chúng tôi đã sử dụng các tính năng tương tự của mục tiêu như công việc trước đây của chúng tôi [148], xem Bảng 5 . Cần lưu ý rằng các trình mô phỏng này không hỗ trợ phản ứng tổng hợp hoạt động vi mô ( *µ* -op), do đó, độ rộng của các giai đoạn đường ống được đặt thành số lượng *µ* -ops đơn giản có thể so sánh được (ví dụ: chiều rộng vấn đề là bốn hoạt động hợp nhất, được đặt là sáu thao tác đơn giản cho các thí nghiệm của chúng tôi).

Chúng tôi đã định cấu hình trình mô phỏng để phù hợp với hệ thống tham chiếu đã chọn theo hiểu biết tốt nhất của chúng tôi. Chúng tôi cũng đảm bảo rằng cấu hình trên các trình mô phỏng tương tự nhau, chỉ có những thay đổi tối thiểu đối với mã của trình mô phỏng để có thể thực hiện so sánh công bằng dựa trên kết quả mô phỏng. Mục đích của nghiên cứu này là so sánh độ chính xác tuyệt đối và tương đối của các trình mô phỏng với nhau chứ không phải để xác nhận các trình mô phỏng. Vì vậy, chúng tôi đã không cố gắng triển khai các cấu trúc hoặc tối ưu hóa phần cứng mới mà một số trình mô phỏng nhất định không hỗ trợ. Bảng 6 mô tả một số tham số cấu hình không giống nhau trên tất cả các trình mô phỏng hoặc khác với cấu hình Haswell do những hạn chế về hỗ trợ của trình mô phỏng [162]. Ví dụ: các thông số kỹ thuật của bộ dự đoán nhánh Haswell chưa được biết và các bộ dự đoán nhánh được hỗ trợ trong các trình mô phỏng được nghiên cứu không hoàn toàn giống nhau. Như được hiển thị trong Bảng 6 , chúng tôi đã định cấu hình các bộ dự đoán nhánh càng gần nhau càng tốt. Công cụ dự đoán nhánh giải đấu của gem5 dựa trên máy Alpha 21264 [163] và sử dụng bảng lịch sử cục bộ và toàn cầu cùng với công cụ dự đoán lựa chọn. Sniper sử dụng bộ dự đoán nhánh được mô phỏng theo đơn vị dự đoán nhánh của Intel Pentium M. Công cụ dự đoán này giống hệt với công cụ dự đoán BLG nối tiếp của McFarling [164] và cũng sử dụng các công cụ dự đoán vòng lặp, lưỡng kim và toàn cầu [165]. Để hỗ trợ điều này, chúng tôi đã thay đổi một số cấu hình được mã hóa cứng trong Sniper. Sniper cũng chứa bộ đệm mục tiêu nhánh gián tiếp gồm 256 mục ( iBTB ) [165]. PTLsim có nhiều lựa chọn liên quan đến cấu hình dự đoán nhánh. Bộ dự đoán lưỡng kim kết hợp và G-share được định cấu hình cho các thử nghiệm này. Bộ dự đoán nhánh tương tự cũng được sử dụng cho MARSSx86. Công cụ dự đoán nhánh giải đấu chứa công cụ dự đoán lưỡng kim và hai cấp độ, được định cấu hình cho Multi2sim. Kích thước của các yếu tố dự đoán riêng lẻ được thể hiện trong Bảng 6 . PTLsim hỗ trợ cả hàng đợi vấn đề hướng dẫn được phân vùng và chia sẻ [148].

PTLsim không cho phép mô hình hóa các hàng đợi và cụm lệnh dùng chung cùng một lúc. Do đó, theo ví dụ của [166], chúng tôi đã cấu hình các hàng đợi lệnh được phân vùng trong PTLsim với các mục bổ sung để giải quyết mọi tổn thất về hiệu suất do các hàng đợi được phân vùng gây ra. Haswell không biết chi tiết về cấu trúc tìm nạp trước bộ đệm, vì vậy chúng tôi đã tắt tính năng tìm nạp trước trên phần cứng thực và cũng không định cấu hình chúng trên trình mô phỏng. Bạn có thể tìm thấy chi tiết về tất cả các cấu hình được sử dụng cho tất cả các trình mô phỏng trong báo cáo kỹ thuật của chúng tôi [162].

*B. KHỐI LƯỢNG CÔNG VIỆC THỬ NGHIỆM VÀ HIỆU SUẤT*

*ĐO LƯỜNG TRÊN PHẦN CỨNG THẬT*

Chúng tôi đã sử dụng SPEC-CPU2006 [167] và một tập hợp con của điểm chuẩn nhúng MiBench [168]. Các điểm chuẩn được nhúng có thể hoàn thành quá trình thực thi của chúng trong thời gian thực tế trên hệ thống mô phỏng, nhưng việc thực thi hoàn toàn các điểm chuẩn SPEC có thể mất một thời gian rất dài. Do đó, chúng tôi đã chạy từng điểm chuẩn SPEC cho 500 triệu lệnh x86. Các hướng dẫn này được chọn từ một phân đoạn đại diện của chương trình sử dụng Simpoint [125]. Ngoài ra, thời gian khởi động gồm 100 triệu lệnh đã được sử dụng.

Trên phần cứng thực, chúng tôi đã sử dụng PAPI [169] để đo các hướng dẫn trên mỗi chu kỳ (IPC), lỗi bộ đệm, các giá trị dự đoán sai nhánh cho toàn bộ quá trình thực thi các điểm chuẩn được nhúng. Đối với các điểm chuẩn SPEC-CPU2006, chúng tôi đã đo các tham số tương tự cho cùng 500 triệu lệnh mô phỏng. Để loại bỏ ảnh hưởng của nhiễu loạn hệ thống đối với các phép đo sự kiện phần cứng thực, chúng tôi đã chạy điểm chuẩn nhiều lần theo cách không liên tục rồi tính giá trị trung bình của tất cả các lần chạy. Độ lệch chuẩn trong kết quả IPC cho tất cả các lần chạy trung bình là 0,02265. Chúng tôi đã sử dụng trình biên dịch gcc 4.4.7 để biên dịch các điểm chuẩn khác nhau. Chúng tôi đã tạo cả hai tệp nhị phân 32 bit và 64 bit của điểm chuẩn cho các thử nghiệm của mình (tùy thuộc vào những gì mỗi trình mô phỏng hỗ trợ). Cụ thể, chúng tôi đã sử dụng các tệp nhị phân 32-bit cho Multi2Sim, PTLsim và Sniper (cũng đã sử dụng các tệp nhị phân 64-bit với Sniper), trong khi chúng tôi sử dụng các tệp nhị phân 64-bit cho các trình mô phỏng còn lại. Các tệp nhị phân tương tự đã được sử dụng cho các trình mô phỏng và chạy phần cứng thực. Hệ điều hành máy chủ được sử dụng để xây dựng và chạy khối lượng công việc lần lượt là Scientific Linux 2.6.32 (32 bit) và Ubuntu 14.04 (64 bit) cho các tệp nhị phân 32 bit và 64 bit. Các giá trị bộ đếm hiệu suất của phần cứng tham chiếu đã được tính toán trên cả hai máy chủ. Tác phẩm này sử dụng phiên bản ổn định của gem5 tháng 9 năm 2015, phiên bản MARSSx86 0.4, phiên bản Multi2Sim 5.0, phiên bản Sniper 6.0, phiên bản ổn định của ZSim tháng 4 năm 2016 và phiên bản PTLsim có sẵn tại [170] cho tất cả các thử nghiệm.

**IX. KẾT QUẢ**

Chúng tôi đã mô phỏng các điểm chuẩn được đề cập trước đó trên sáu trình mô phỏng khác nhau và so sánh kết quả mô phỏng của chúng với kết quả phần cứng mục tiêu thực. Chúng tôi đã tính toán sai số thử nghiệm của trình mô phỏng so với các lần chạy phần cứng thực. Chúng tôi cũng đã thực hiện kiểm tra độ nhạy để tìm ra tác động của việc thay đổi các tham số cấu hình nhất định của từng trình mô phỏng so với các trình mô phỏng khác.

*A. PHÂN TÍCH LỖI*

Hình 5 và Hình 6 hiển thị lỗi phần trăm trong các giá trị IPC cho tất cả các điểm chuẩn trên tất cả các trình mô phỏng khi so sánh với các giá trị IPC của các điểm chuẩn từ các lần chạy phần cứng tham chiếu. Sai số phần trăm tuyệt đối trung bình (MAPE) trong các giá trị IPC là thấp nhất đối với Sniper đối với tất cả các loại điểm chuẩn.

Để nghiên cứu các nguồn lỗi gây ra sự thiếu chính xác quan sát được trong các giá trị IPC, chúng tôi đã xem xét các lỗi bộ đệm và dự đoán sai nhánh đối với điểm chuẩn SPEC trên các trình mô phỏng này. Hình 7 và 8 hiển thị tỷ lệ phần trăm lỗi trong bộ đệm dữ liệu L1 bị lỗi, Hình 9 và 10 hiển thị tỷ lệ phần trăm lỗi trong bộ đệm L3 bị lỗi và Hình 11 và 12 hiển thị tỷ lệ phần trăm lỗi trong số nhánh có điều kiện bị dự đoán sai được hiển thị bởi các trình mô phỏng khác nhau. Tỷ lệ phần trăm sai số trong các thống kê này là rất cao đối với một số điểm chuẩn (cao hơn nhiều so với 100%) như trong hình. Trung bình, điểm chuẩn FP-SPEC cho thấy tỷ lệ lỗi cao hơn khi bỏ sót bộ đệm so với điểm chuẩn INT-SPEC vì chúng bao gồm số lượng lệnh bộ nhớ lớn hơn và điểm chuẩn INT-SPEC cho thấy tỷ lệ lỗi cao hơn về độ chính xác dự đoán nhánh vì chúng bao gồm số lượng lệnh lớn hơn hướng dẫn chi nhánh.

PTLsim cho thấy độ chính xác cao trong các điểm chuẩn dấu phẩy động; một số điểm chuẩn cho thấy độ chính xác trên 50%. Lý do chính mà chúng tôi nhận thấy dẫn đến việc đánh giá thấp IPC trong PTLsim là liên quan đến việc giải mã các lệnh x86 thành *µ* -ops. Các điểm chuẩn hiển thị các giá trị IPC có độ chính xác cao, thể hiện tỷ lệ cao giữa các hướng dẫn *µ* -ops và x86 cho PTLsim so với các trình mô phỏng khác. Ví dụ: tỷ lệ này cho *gemsFDTD* , *gamess* , *povray* và *soplex* lần lượt là 9, 6,07, 5,43 và 4,2 trên PTLSim . Trên gem5 và Multi2Sim, tỷ lệ lệnh *µ* -ops trên x86 được quan sát cho cùng một điểm chuẩn luôn nhỏ hơn 2,30. Vì chiều rộng đường ống trong các trình mô phỏng này được xác định bởi số lượng *µ* -ops nên giá trị cao của tỷ lệ lệnh *µ* -ops trên x86 sẽ ảnh hưởng đến hiệu suất của đường ống.

Có nhiều ví dụ trong Hình 7 và 12, cho thấy độ chính xác hơn 100% tương ứng trong các lỗi bộ nhớ đệm dữ liệu L1 và dự đoán sai nhánh. Những số liệu này giúp hiểu được các lỗi tiêu cực về số IPC đối với một số ứng dụng. Ví dụ: nhiều điểm chuẩn chạy trên gem5 ( *gobmk* , *gcc* \_200, *h* 264 *ref* , *perlbench* , *povray* , *namd* ) thể hiện tỷ lệ dự đoán sai nhánh được đánh giá quá cao và lỗi bộ đệm dữ liệu so với kiến trúc mục tiêu tham chiếu. Đối với các điểm chuẩn này, bộ dự đoán nhánh của trình mô phỏng không mô phỏng hành vi của bộ dự đoán nhánh của lõi thực tế, dẫn đến độ chính xác cao hơn. Khi các điểm chuẩn có số lượng lỗi dự đoán nhánh rất cao được so sánh với các điểm chuẩn có số lượng lỗi dự đoán nhánh thấp hơn, người ta nhận thấy rằng chúng chứa số lượng lệnh nhánh cao hơn nhiều (20% hoặc nhiều hơn trong hỗn hợp lệnh tổng thể) . Số lượng cao này cho thấy sự thiếu sót của bộ dự đoán nhánh của trình mô phỏng để lập mô hình bộ dự đoán nhánh của mục tiêu (Haswell) cho các điểm chuẩn đó.

Cách giải mã và triển khai một số lệnh x86 của gem5 có thể giải thích một số điểm không chính xác của IPC. Một ví dụ là phép chia số nguyên, được giải mã thành nhiều *µ* -op, trong đó mỗi *µ* -op chịu trách nhiệm tính một bit thương. Thuật toán phân chia phần cứng trên hệ thống thực khác với cách triển khai của trình mô phỏng. Hơn nữa, *µ* -ops bị gắn nhãn sai trong gem5 gây ra một số kết quả không chính xác [15]. Một vấn đề khác được quan sát thấy với gem5 liên quan đến thông lượng của giai đoạn tìm nạp của nó. Việc triển khai giai đoạn tìm nạp hiện tại không cho phép bắt đầu các yêu cầu mới khi nó đang chờ phản hồi, ví dụ: lỗi bộ đệm lệnh. Kết quả là, đơn vị tìm nạp không được hưởng lợi từ bộ đệm lệnh không chặn khi độ trễ truy cập bộ đệm lệnh lớn hơn một chu kỳ.

Tương tự, hầu hết các điểm chuẩn cho thấy độ chính xác cao trên Multi2Sim, ví dụ *gcc* \_200, *hmmer* , *h* 264 *ref* , *gemsFDTD* , *namd* và *Perlbench* . cũng cho thấy dự đoán nhánh cao bị bỏ lỡ. Những sai sót này có tác động đáng kể đến kết quả hoạt động tổng thể. Giống như gem5, nhiều điểm chuẩn trong số này chứa số lượng lớn các hoạt động nhánh (20% trở lên trong hỗn hợp hướng dẫn tổng thể). Các trình mô phỏng khác cũng hiển thị số lượng phóng đại của bộ dự đoán nhánh và lỗi bộ đệm trong trường hợp IPC không chính xác cao hơn, ví dụ: *h* 264 *ref* , *libquantum* , *milc* và *povray* trên MARSSx86 và *libquantum* và *mcf* trên ZSim .

Đáng chú ý là các giá trị IPC bị đánh giá thấp không phải lúc nào cũng được giải thích bằng những dự đoán sai về nhánh được đánh giá quá cao đã nói ở trên. Ví dụ: mặc dù Sniper hiển thị các dự đoán sai nhánh được đánh giá quá cao đối với *gromacs* , nhưng số lượng dự đoán sai này thực tế là thấp và không có tác động đáng kể đến kết quả hiệu suất. Sự tích tụ độ thiếu chính xác của các lần bỏ lỡ bộ nhớ đệm cấp cao hơn thành các lần bỏ lỡ bộ nhớ đệm cấp thấp hơn giải thích sự thiếu chính xác cao hơn về số lần bỏ lỡ bộ nhớ đệm cấp cuối cùng. Hơn nữa, việc không thể mô phỏng phản ứng tổng hợp *µ* -op và bộ đệm *µ* -op (được sử dụng trong Haswell) trong các trình mô phỏng được nghiên cứu có thể gây ra nhiều sai số không chính xác. Việc thiếu các chi tiết kiến trúc vi mô khác trong các mô hình mô phỏng và các tùy chọn cấu hình lại linh hoạt ngoài mức độ trừu tượng của một số trình mô phỏng cũng có thể tạo ra lỗi trong kết quả mô phỏng.

*B. KIỂM TRA ĐỘ NHẠY*

Một số nghiên cứu hiện tại dựa vào những cải tiến hiệu suất tương đối được thông báo bởi các trình mô phỏng để thiết kế và so sánh các kiến trúc và ý tưởng mới. Để nghiên cứu và so sánh hiệu suất tương đối của trình mô phỏng, chúng tôi đã thực hiện ba thử nghiệm hiệu suất tương đối: (1) thay đổi độ rộng của các giai đoạn đường ống thành một nửa giá trị bình thường của chúng, (2) giảm kích thước của tất cả các bộ đệm xuống một nửa kích thước của chúng từ Bảng 5 và (3) định cấu hình bộ dự đoán nhánh lưỡng kim thay vì bộ dự đoán nhánh được sử dụng cho Haswell. Hình 13 - 18 cho thấy những thay đổi về giá trị IPC (so với IPC mục tiêu Haswell mô phỏng) cho những lần chạy này. Tuy nhiên, thật khó để đánh giá tác động lên IPC sẽ liên quan đến cấu hình cơ sở như thế nào nếu không có thử nghiệm bổ sung; vài quan sát có thể được thực hiện. Ví dụ: các điểm chuẩn chuyên sâu về bộ nhớ (như *mcf ; gemsFDTD ; xalancbmk* ) dự kiến sẽ cho thấy sự thay đổi lớn hơn về hiệu suất tương đối đối với kích thước bộ đệm giảm so với các điểm chuẩn khác. MARSSx86 và Sniper dường như là những trình mô phỏng nhạy cảm nhất đối với việc thay đổi kích thước bộ đệm. Tuy nhiên, MARSSx86 hiển thị hành vi không nhất quán khi thay đổi kích thước bộ đệm cho các điểm chuẩn khác (ví dụ *sjeng* \_ *ref* và *namd* ), so với các trình mô phỏng khác. Nhìn chung, Sniper và gem5 dường như nhạy cảm hơn các trình mô phỏng khác đối với hầu hết các thay đổi trong hầu hết các trường hợp, tùy thuộc vào độ nhạy của điểm chuẩn đối với thay đổi được đề cập. Zsim dường như ít bị ảnh hưởng hơn bởi sự thay đổi độ rộng đường ống giữa tất cả các trình mô phỏng. Mặc dù khó có thể đánh giá hiệu suất tương đối của các trình mô phỏng và đánh giá độ chính xác tương đối của chúng, nhưng các số liệu cho thấy rõ ràng rằng hiệu suất tương đối của các trình mô phỏng khác nhau và sự khác biệt có thể đáng kể trong một số trường hợp.

Ngoài lỗi thử nghiệm lõi đơn và các thử nghiệm hiệu suất tương đối, chúng tôi đã so sánh các trình mô phỏng để tìm lỗi thử nghiệm đa lõi. Hình 19 và 20 hiển thị các giá trị IPC được chuẩn hóa cho lõi kép và lõi tứ chạy cho gem5, MARSSx86, Sniper (nhị phân 64 bit) và ZSim , cho đến kết quả phần cứng thực tế. Chúng tôi đã sử dụng khối lượng công việc đa chương trình từ các điểm chuẩn SPEC CPU2006 làm đầu vào cho các mô phỏng đa lõi. Tuy nhiên, việc sử dụng điểm chuẩn đa luồng cũng rất thú vị; việc thiết lập các điểm chuẩn đó cho tất cả các trình mô phỏng không hề đơn giản và tốn thời gian. Các kết hợp đầu vào lõi kép và lõi tứ của chúng tôi đã được chọn từ điểm chuẩn CPU2006 bằng cách sử dụng trình tạo số ngẫu nhiên. Tỷ lệ lỗi trung bình trung bình cao hơn khi chạy đa lõi so với chạy lõi đơn. Sniper cho thấy độ chính xác cao nhất khi chạy lõi kép và ZSim cho thấy độ chính xác cao nhất khi chạy lõi tứ.

*C. TỐC ĐỘ MÔ PHỎNG*

Cuối cùng, chúng tôi đã đo tốc độ của từng trình mô phỏng và thời gian cần thiết để tua đi nhanh các mô phỏng. Bảng 7 cho thấy thời gian mô phỏng trung bình của các trình mô phỏng đối với từng loại điểm chuẩn. ZSim là trình mô phỏng nhanh nhất trong số tất cả các trình mô phỏng được nghiên cứu. Bảng này cũng hiển thị thời gian chuyển tiếp nhanh của tất cả các trình mô phỏng khi mô phỏng điểm chuẩn CPU SPEC.

*D. TÓM TẮT CÁC QUAN SÁT*

Dưới đây là bản tóm tắt các quan sát chính của chúng tôi dựa trên khảo sát của chúng tôi về các trình mô phỏng khác nhau và nghiên cứu lỗi thử nghiệm.

• Các thử nghiệm chỉ ra mối tương quan chặt chẽ giữa độ chính xác của trình mô phỏng so với phần cứng thực và sự tồn tại của quá trình xác nhận và hiệu chuẩn của trình mô phỏng cho kiến trúc mục tiêu tương ứng. Ví dụ: Sniper và ZSim , có ít lỗi nhất trong các thử nghiệm của chúng tôi, đã được xác thực và hiệu chỉnh cho lõi Intel Nehalem và Westmere tương ứng [23], [143].

• Kết quả của các trình mô phỏng không được hiệu chỉnh/không hợp lệ có thể khác biệt đáng kể so với các con số hiệu suất phần cứng tham chiếu. Điều này cũng được quan sát bởi Asri et al. khi hiệu chỉnh MARSSx86 cho một máy mục tiêu cụ thể [140].

• Bộ dự đoán nhánh và lỗi bộ đệm có độ chính xác cao, hướng dẫn giải mã *µ* -ops không chính xác và sự vắng mặt của một số cấu trúc tối ưu hóa Haswell trong các trình mô phỏng được nghiên cứu là những nguyên nhân chính dẫn đến kết quả mô phỏng không chính xác.

• Độ chính xác tương đối của trình mô phỏng có thể thay đổi đáng kể, điều này có thể dẫn đến kết luận không chính xác.

• Trình mô phỏng chính xác hơn có thể vẫn không đáp ứng được nhu cầu của bạn. Ví dụ: Sniper và ZSim không linh hoạt lắm trong việc mô hình hóa các tính năng kiến trúc vi mô mới so với gem5 và MARSSx86, mặc dù chúng cho thấy độ chính xác cao hơn. Mặt khác, gem5 và MARSSx86 thể hiện tính linh hoạt hơn, hỗ trợ thử nghiệm các ý tưởng thiết kế vi kiến trúc mới và nghiên cứu hiệu suất của các khối kiến trúc vi mô cụ thể.

• Sniper và ZSim là những lựa chọn hấp dẫn để mô phỏng kiến trúc x86 nhiều lõi vì chúng nhanh hơn và tạo ra độ chính xác cao hơn các trình mô phỏng khác.

• gem5 và MARSSx86 hỗ trợ mô phỏng toàn hệ thống và có thể được sử dụng để mô phỏng các ứng dụng có hệ thống

cuộc gọi, để có độ chính xác cao hơn khi ứng dụng chứa nhiều cuộc gọi hệ thống. Chúng cũng có thể được sử dụng để nghiên cứu ảnh hưởng của tương tác hệ điều hành với phần cứng đến hiệu suất của ứng dụng. ZSim và Sniper là các trình mô phỏng cấp độ ứng dụng .

• Độ chính xác hợp lý, mô hình mô phỏng chi tiết, hỗ trợ hệ thống đầy đủ và tốc độ tốt của MARSSx86 khiến nó trở thành một lựa chọn tốt cho các nghiên cứu hệ thống đầy đủ chi tiết, đặc biệt là cho các mục tiêu đa lõi. PTLsim có thể đóng vai trò là nền tảng để xây dựng các trình mô phỏng nâng cao hơn, ví dụ MARSSx86 [4].

• Một trường hợp sử dụng quan trọng của Multi2Sim là mô phỏng kiến trúc CPU-GPU.

• Khả năng cấu hình của gem5, hỗ trợ nhiều ISA khác nhau và hỗ trợ một hệ điều hành hoàn chỉnh cùng với cộng đồng các nhà phát triển tích cực của nó, khiến gem5 trở thành lựa chọn thuyết phục để chạy các thử nghiệm toàn diện trên một khối bộ xử lý cụ thể hoặc toàn bộ lõi, để nghiên cứu sự tương tác giữa phần cứng và hệ điều hành hoặc để nghiên cứu các hệ thống không đồng nhất (ISA đơn, đa ISA, CPU-GPU, CPU-ASIC).

**X. KẾT LUẬN VÀ ĐỊNH HƯỚNG TƯƠNG LAI**

Bài báo này trình bày một nghiên cứu chi tiết về mô phỏng kiến trúc máy tính. Chúng tôi đã thực hiện một cuộc khảo sát so sánh các trình mô phỏng khác nhau và phân loại chúng thành các loại khác nhau. Chúng tôi đã kiểm tra độ chính xác tuyệt đối và tương đối của sáu trình mô phỏng kiến trúc máy tính hiện đại bằng cách so sánh kết quả mô phỏng của chúng với kết quả của phần cứng thực. Kết quả thử nghiệm cho thấy Sniper tạo ra sai số tuyệt đối tối thiểu và ZSim là tốc độ nhanh nhất cho mô phỏng lõi đơn. Tuy nhiên, việc chọn một trình mô phỏng để sử dụng còn tùy thuộc vào mục đích học tập/nghiên cứu và các tính năng của trình mô phỏng. Công trình này cũng nhấn mạnh tầm quan trọng của việc xác nhận các trình mô phỏng và chỉ ra một số nguyên nhân gây ra sự thiếu chính xác trong các trình mô phỏng kiến trúc máy tính.

Với sự xuất hiện của kiến trúc nhiều lõi, ứng dụng mới và các tùy chọn thiết kế không đồng nhất, nhu cầu về các kỹ thuật tăng tốc mô phỏng mới và cải tiến mà không làm giảm độ chính xác của trình mô phỏng ngày càng tăng. Ngoài ra, việc tạo các trình mô phỏng mô-đun và có tài liệu đầy đủ cập nhật giúp trình mô phỏng linh hoạt hơn và dễ sử dụng hơn. Việc tích hợp các thành phần không đồng nhất hơn trong các trình mô phỏng kiến trúc bộ xử lý cho phép thực hiện các nghiên cứu và hướng nghiên cứu mới, đặc biệt khi sự tương tác giữa lõi bộ xử lý với các thành phần không đồng nhất và bộ tăng tốc được mô hình hóa linh hoạt. Hơn nữa, khi định luật Moore sắp kết thúc, những đổi mới về kiến trúc trong tương lai sẽ yêu cầu tối ưu hóa trên toàn bộ hệ thống điện toán phần cứng/phần mềm. Các trình mô phỏng kiến trúc hiện tại không được thiết kế để nghiên cứu các tối ưu hóa nhiều lớp như vậy. Vì vậy, nhu cầu cấp thiết là phải đưa ra các kỹ thuật mô phỏng mới cho phép các nhà nghiên cứu thử nghiệm ý tưởng của họ trên toàn bộ hệ thống.

Cuối cùng, việc xác nhận mô phỏng là rất quan trọng để đảm bảo độ tin cậy trong kết quả được tạo ra và tạo điều kiện cho khả năng tái tạo. Mặc dù nhiều nghiên cứu dựa vào hiệu suất tương đối của trình mô phỏng, nhưng thử nghiệm của chúng tôi cho thấy rằng các trình mô phỏng khác nhau cho thấy hiệu suất tương đối khác nhau và sự khác biệt có thể rất đáng kể. Vì vậy, cần có các phương pháp mới để xác nhận độ chính xác tương đối của trình mô phỏng. Tương tự, cần phải xác thực các trình mô phỏng cho mô phỏng bộ xử lý đa lõi, vì sự thiếu chính xác của mô phỏng đa lõi đối với các trình mô phỏng đã được xác thực cho các bộ xử lý lõi đơn cho thấy tỷ lệ lỗi cao đối với các thử nghiệm đa lõi.