ITRC
ICT Research Institute
(Iran Telecommunication Research Center)

Journal of Information and Communication Technology

Iranian Society of Information and Communication Technology

Volume 17, Issue 63-64, Spring and Summer 2025, pp. 204-218

Online Estimation and Improvement of Cache Soft Error Vulnerability

Mohammad Moeini Jahromi[1], Mohammad Hasan Ahmadilivani[2], Mostafa E.Salehi Nasab[3]*

[1]School of Electrical and Computer Engineering, Engineering Faculty, University of Tehran, Tehran, Iran
[2]School of Electrical and Computer Engineering, Engineering Faculty, University of Tehran, Tehran, Iran
[3]School of Electrical and Computer Engineering, Engineering Faculty, University of Tehran, Tehran, Iran

Received: 10 November 2024, Revised: 19 February 2025, Accepted: 17 March 2025
Paper type: Research

Abstract

Due to the high density of transistors, memories are highly susceptible to soft errors. The processor's cache, by holding execution data and having frequent interactions with it, greatly impacts system reliability. This importance is even higher in embedded systems and safety-critical applications. One of the most significant factors affecting the reliability of the cache is its size. Smaller caches have better reliability due to their smaller area and shorter data retention, but reducing the cache size makes program execution times longer. This increases the probability of a soft error. Furthermore, reliability of cache is not uniform during program execution, and fixed size of memory cannot optimize its reliability during this time. In this regard, the main issue in improving cache vulnerability is to determine an optimum size of cache and its change time according to change overhead. Accordingly, this paper defines a model for estimating cache vulnerability, which determines vulnerability based on cache data and the type of access to it. Based on the proposed model, an algorithm has been implemented that estimates cache vulnerability online during execution. To model time in this approach, counters are used that model access times during decision-making intervals. By estimating based on blocks instead of memory words and determining the sizes of the counters and decision intervals, the proposed method has been optimized. The accuracy of the vulnerability trend estimation compared to the reference model is 95.22%. Additionally, by using the estimated vulnerability trend during execution and the effective cache size of each program, an algorithm for reconfiguring the cache to improve its vulnerability has been proposed. Implementation showed that with only 5.4% area overhead and 6% time overhead, we can have a reconfigurable memory equipped with a vulnerability management algorithm, which has a lower runtime vulnerability than a fixed cache size and overall vulnerability improvement of 36%.

Keywords: Reliability, Soft Error, Error Masking, Cache, Cache Vulnerability, Reliability-Performance Trade-off, Cache Size, Online Cache Vulnerability Estimation.

* Corresponding Author's email: mersali@ut.ac.ir

دوفصلنامه
فناوری اطلاعات و ارتباطات ایران

سال هفدهم، شماره‌های ۶۳ و ۶۴ ، بهار و تابستان ۱۴۰۴، صفحه ۲۰۴ الی ۲۱۸

تخمین و بهبود آنلاین آسیب پذیری خطای نرم حافظه نهان

محمد معینی جهرمی[۱]، محمد حسن احمدی لیوانی[۲]، مصطفی ارسالی صالحی نسب[۳]*

[۱] دانشجو مقطع دکترا، دانشکده مهندسی برق و کامپیوتر، دانشکدگان فنی، دانشگاه تهران، تهران، ایران

[۲]کارشناسی ارشد، دانشکده مهندسی برق و کامپیوتر، دانشکدگان فنی، دانشگاه تهران، تهران، ایران

[۳] استادیار، دانشکده مهندسی برق و کامپیوتر، دانشکدگان فنی، دانشگاه تهران، تهران، ایران

تاریخ دریافت: ۱۴۰۳/۰۸/۲۰ تاریخ بازبینی: ۱۴۰۳/۱۲/۰۱ تاریخ پذیرش: ۱۴۰۳/۱۲/۲۷

نوع مقاله: پژوهشی

چکیده

حافظه‌ها به دلیل چگالی بالای ترانزیستورها در آن‌ها به شدت در معرض خطاهای نرم قرار دارند. حافظه نهان پردازنده به دلیل نگه داشتن اطلاعات اجرایی و تعاملات زیاد با آن، قابلیت اطمینان سیستم را به شدت تحت تأثیر قرار می‌دهد. در سیستم‌های نهفته و کاربردهای ایمنی-بحرانی، اهمیت آن به مراتب بیشتر می‌شود. از مهم‌ترین پارامترهای تأثیرگذار بر قابلیت اطمینان حافظه نهان، حجم آن است. حافظه نهان با حجم کم‌تر، به واسطه مساحت کوچک‌تر و ماندگاری کم‌تر داده‌ها در آن قابلیت اطمینان بیشتری دارد اما، کاهش حجم حافظه نهان، مدت اجرای برنامه‌ها را طولانی‌تر می‌کند. این افزایش زمان اجرای برنامه‌ها، احتمال بروز خطای نرم را افزایش می‌دهد. از سویی، قابلیت اطمینان حافظه نهان در طول اجرای یک برنامه یکنواخت نیست و ثابت بودن حجم حافظه نمی‌تواند قابلیت اطمینان آن را در طول اجرا بهینه کند. در این راستا، مسأله اصلی در بهبود آسیب‌پذیری حافظه نهان، تعیین اندازه حافظه نهان و زمان تغییر آن با توجه به سربار تغییرات است. بر همین مبنا، در این مقاله مدلی برای تخمین آسیب‌پذیری حافظه نهان تعریف شده است که بر اساس داده‌های حافظه نهان و نوع دسترسی به آنها، آسیب‌پذیری آن تعیین می‌شود. بر اساس مدل ارائه شده، الگوریتمی پیاده‌سازی شده است که آسیب‌پذیری حافظه نهان را در زمان اجرا به صورت آنلاین تخمین می‌زند. برای مدل‌سازی زمان در این روش، از شمارنده‌هایی استفاده شده است که در طول بازه‌های تصمیم‌گیری، زمان دسترسی‌ها را مدل می‌کنند. با استفاده از تخمین بلوک بجای کلمات حافظه و تعیین اندازه شمارنده‌ها و بازه‌های تصمیم‌گیری، روش ارائه شده، بهینه‌سازی شده است. دقت تخمین روند آسیب‌پذیری نسبت به مدل رفرنس، ۹۵/۲۲٪ می‌باشد. همچنین با استفاده از تخمین روند آسیب‌پذیری در زمان اجرا و اندازه موثر حافظه نهان هر برنامه، الگوریتمی جهت بازپیکربندی حافظه نهان در جهت بهبود آسیب‌پذیری آن ارائه شده است. پیاده‌سازی این طراحی نشان داده است که تنها با سربار مساحت ٪۵/۴ و سربار زمانی ٪۶ می‌توان یک حافظه با قابلیت بازپیکربندی و مجهز به الگوریتم مدیریت آسیب‌پذیری داشت که آسیب‌پذیری آن در زمان اجرا از آسیب‌پذیری حافظه نهان با حجم ثابت کم‌تر و آسیب‌پذیری کل آن نیز ٪۳۶ بهتر باشد.

کلیدواژگان: قابلیت اطمینان، خطای نرم، پوشش خطاها، حافظه نهان، آسیب‌پذیری حافظه نهان، مصالحه قابلیت اطمینان و کارآیی، حجم حافظه نهان، تخمین آسیب‌پذیری در زمان اجرا.

* رایانامهٔ نویسنده مسؤول: mersali@ut.ac.ir

۱- مقدمه

پیشرفت تکنولوژی ساخت ترانزیستورها و کاهش ابعاد آن‌ها بر اساس قانون مور، علاوه بر افزایش کارآیی و سرعت پردازش پردازنده‌ها، محدودیت‌ها و مخاطراتی را از قبیل نقطه‌ی داغ، افزایش توان و انرژی مصرفی، محدودیت ولتاژ و فرکانس، افزایش آسیب‌پذیری تراشه‌ها و بروز اشکال و خطا برای آن‌ها به همراه آورده است [۱]. از جمله خطراتی که یک سیستم را خصوصاً در کاربردهای بحرانی-ایمنی[1] تهدید می‌کند و می‌تواند منجر به عملکرد نادرست آن شود، بروز اشکال در منطق محاسباتی و یا داده‌های ذخیره شده سیستم است. برخورد ذرات پرانرژی به قطعات الکترونیکی و القای انرژی به آن‌ها، ممکن است در مسیر محاسبات منطقی، اشکال ایجاد کند و یا تغییر مقادیر ذخیره شده را در پی داشته باشد و در نهایت خروجی را دستخوش تغییر کند [۲]. بروز چنین اشکالاتی که به آن‌ها خطای نرم[2] گفته می‌شود، ممکن است منجر به ایجاد خروجی اشتباه در سیستم شود که امری نامطلوب است.

قابلیت اطمینان[3]، احتمال کار کردن صحیح یک سیستم در بازه زمانی مشخصی را بیان می‌دارد. با روند پیش‌گفته در پیشرفت تکنولوژی، طراحی سیستم‌ها با در نظرگیری قابلیت اطمینان، اهمیت بالایی پیدا کرده است. صحت داده‌های پردازش شده در یک پردازنده، تأثیر مستقیم بر عملکرد صحیح آن از دید کاربر دارد. لذا اهمیت و تأثیر حفاظت از صحت داده‌ها بر قابلیت اطمینان پردازنده واضح است. حافظه نهان پردازنده از جمله عناصر یک سیستم روی تراشه است که به دلیل مساحت زیاد، تراکم ترانزیستورها و ساخته شدن از سلول‌های SRAM، با کاهش ابعاد ترانزیستورها بیش از پیش در معرض خطاهای نرم قرار دارد و ارائه طراحی‌های گوناگون برای حفاظت از داده‌های آن متناسب با روند تکنولوژی در حال پیش‌روی است [۳].

از روش‌های رایج محافظت از داده‌های حافظه، استفاده از کدهای تشخیص و تصحیح خطا است [۴،۵] که سربارهای مساحت، انرژی و کارآیی به سیستم تحمیل می‌کند و استفاده از آن‌ها در حافظه نهان با محدودیت‌های جدی روبه‌روست. به عنوان مثال، تکنیک ساده تصحیح خطای تکی و تشخیص خطای دوگانه (SECDED[4]) نیاز به هفت ستون اضافه برای محافظت از حافظه ۳۲ بیتی دارد که منجر به سربار ٪۲۱/۹ مساحت می‌شود و توان را نیز به صورت نسبی افزایش می‌دهد [۶]. برای کاهش سربارهای ناشی از استفاده کدهای تشخیص و تصحیح خطا، بهینه‌سازی‌هایی در پیاده‌سازی این الگوریتم‌ها انجام شده است [۷-۱۱]. همچنین، تکنیک‌هایی در سطح ریزمعماری برای استفاده کاراتر از این کدها ارائه شده است تا از داده‌های حافظه نهان، به صورت غیریکنواخت و با سربارهای کمتری محافظت شود [۱۲-۱۵]. با این حال، روش‌های متعددی از روش‌های تکرار[5] برای بالا بردن قابلیت اطمینان حافظه استفاده کرده‌اند و تعدادی نیز سعی در کاهش احتمال بروز خطای نرم در حافظه کرده‌اند. در تکنیک‌های آنلاین بهینه‌سازی قابلیت اطمینان در سطح سیستم، آسیب‌پذیری خطای نرم در زمان اجرا تخمین زده می‌شود و اقدامات حفاظتی مانند تکرار کار یا فعال‌سازی انتخابی کدهای تشخیص و تصحیح خطا برای کارهای حیاتی با آسیب‌پذیری نسبتاً بالا، استفاده می‌شود. در [۱۶] ایده‌ای برای تشخیص آنلاین آسیب‌پذیری حافظه و استفاده از کدهای تشخیص چند سطحی ارائه شده است که به دلیل سربار و روش ارائه شده، بیشتر برای حافظه اصلی مناسب است. با این حال، سربارهای ناشی از این کدها بر عملکرد پردازنده پابرجاست و بسیاری از آن‌ها قابلیت پیاده‌سازی در حافظه نهان سطح اول را ندارند.

در روش‌های آنلاین بهینه‌سازی آسیب‌پذیری حافظه نهان، نیاز به تخمین آسیب‌پذیری حافظه نهان است. ضریب آسیب‌پذیری حافظه نهان، معیاری است که از آن برای مدل کردن قابلیت اطمینان حافظه نهان استفاده می‌شود [۱۷]. آسیب‌پذیری حافظه نهان، بر اساس الگوی دسترسی‌ها به این حافظه تعیین می‌شود و داده‌ها بین بازه‌های خواندن و پس‌نویسی در حافظه سطح پایین‌تر، آسیب‌پذیر هستند. لذا نوع برنامه و ساختار حافظه نهان بر این معیار اثرگذار است. کاهش آسیب‌پذیری حافظه نهان، افزایش قابلیت اطمینان آن را در پی دارد و بررسی‌ها درمورد این معیار نشان می‌دهد که حجم حافظه نهان، بیشترین اثرگذاری را بر آسیب‌پذیری دارد [۱۸-۲۰]. حافظه نهان بزرگ‌تر، ضریب آسیب‌پذیری بالاتری دارد اما از طرفی باعث کاهش زمان اجرای برنامه‌ها و کاهش احتمال بروز خطای نرم می‌شود. لذا، بین قابلیت اطمینان حافظه نهان و کارآیی آن یک مصالحه[6] در رابطه با حجم حافظه برقرار است.

با توجه به مصالحه بین مدت زمان اجرای برنامه‌ها و آسیب‌پذیری حافظه نهان، بررسی کاملی جهت انتخاب حجم حافظه نهان و رفتار این دو پارامتر با تغییر حجم حافظه برای برنامه‌های مختلف انجام شده است و از این طریق بر اساس خواص برنامه‌ها، معیاری برای

[1] Safety-critical
[2] Soft Error
[3] Reliability
[4] Single Error-Correction Double Error-Detection
[5] Redundancy
[6] Trade-off

تعیین حجم حافظه نهان ارائه شده است که بر اساس آن رفتار آسیب‌پذیری حافظه نهان برای کاربردهای مختلف مشابه شود [۲۱]. با استفاده از این معیار می‌توان مصالحه‌ای بین قابلیت اطمینان و کارآیی سیستم برقرار کرد.

در کنار مسأله تأثیر حجم حافظه نهان بر کل آسیب‌پذیری آن، آسیب‌پذیری حافظه نهان در طول اجرای یک برنامه نیز غیریکنواخت است [۲۲]. به این معنی که در بخش‌هایی از برنامه به دلیل نوع دسترسی‌ها به آدرس‌های آن، ماندگاری داده‌ها در حافظه نهان به نحوی است که آسیب‌پذیری حافظه بیشتر یا کم‌تر می‌شود. در نتیجه، ثابت بودن حجم حافظه نهان در تمام طول اجرا لزوماً از لحاظ پایین بودن آسیب‌پذیری کارا نیست و کاهش حجم حافظه نهان در شرایط مشخصی، می‌تواند به پایین نگه داشتن آسیب‌پذیری و پوشش خطاهای احتمالی کمک کند. در این راستا، به دست آوردن تخمینی مناسب از وضعیت آسیب‌پذیری حافظه نهان در زمان اجرا، ضروری می‌نماید.

در بیشتر ایده‌های مطرح شده در این راستا، تأثیر پارامترهای مختلف حافظه نهان بر روی آسیب‌پذیری آن به صورت آفلاین بررسی می‌شود و بر اساس پیش‌پردازش انجام شده، در زمان اجرا از نتایج آن استفاده می‌شود [۲۳-۲۵]. مشکل اصلی این روش‌ها، هزینه بالای زمانی و پردازشی محاسبات آفلاین است به طوری که گاهی انجام کامل آن غیر ممکن بوده و قابل انجام نیست. ایده‌هایی از جمله حالت شبه بهینه و استفاده از شبکه عصبی برای کاهش این زمان و قابل انجام بودن این روش‌ها ارائه شده است [۲۵]. با این وجود، در این روش‌ها با تغییر برنامه کاربردی نیاز است تمام پردازش‌های اولیه مجدد انجام شود.

با توجه به تفاوت آسیب‌پذیری حافظه نهان در ارتباط با حجم حافظه، تا کنون تمامی مقالاتی که در این حوزه کار کرده‌اند به بررسی آسیب‌پذیری حافظه نهان با در نظر گیری حجم ثابتی از حافظه برای برنامه‌های مختلف پرداخته‌اند. در حالی که برنامه‌های مختلف، به حجم‌های حافظه متفاوتی برای اجرا نیاز دارند و تخصیص نامتناسب حجم حافظه به آن‌ها سربار انرژی و آسیب‌پذیری به سیستم تحمیل می‌کند. همچنین با توجه به تفاوت رفتار آسیب‌پذیری حافظه در طول اجرای برنامه‌ها و چالش‌های مربوط به تخمین آسیب‌پذیری حافظه نهان در زمان اجرا، تا کنون روشی مبتنی بر تخمین میزان آسیب‌پذیری حافظه نهان در زمان اجرا و تغییر حجم حافظه نهان بر مبنای آن برای بهبود آسیب‌پذیری یا مدت زمان اجرای برنامه‌ها ارائه نشده است. در این مقاله الگوریتمی جهت بهبود آسیب‌پذیری حافظه نهان ارائه و پیاده‌سازی شده است که با تغییر حجم حافظه نهان در زمان اجرا، احتمال بروز خطای نرم را کاهش می‌دهد.

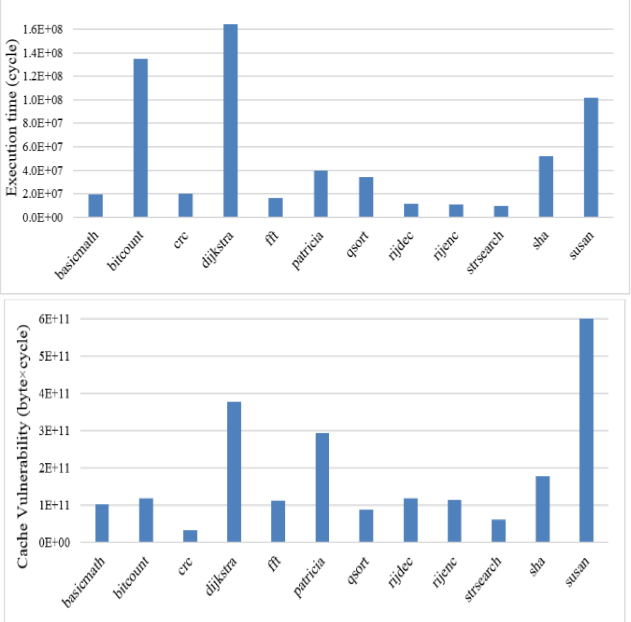
نوآوری‌های مقاله را به صورت زیر می‌توان بیان کرد:

- ارائه مدلی برای محاسبه آسیب‌پذیری حافظه نهان
- تخمین آسیب‌پذیری حافظه نهان در زمان اجرا
- استفاده از تغییر حجم حافظه نهان برای بهبود آسیب‌پذیری آن
- استفاده از اندازه موثر هر برنامه در جهت بهبود آسیب‌پذیری حافظه نهان
- ارائه الگوریتمی برای تخمین و تغییر حافظه نهان در جهت بهبود آسیب‌پذیری حافظه نهان

ساختار این مقاله به شرح زیر سازماندهی‌شده است: در بخش دو به انگیزه‌ای که باعث شد این تحقیق انجام شود خواهیم پرداخت و در بخش سوم به بیان کارهایی که در پژوهش‌های پیشین در این حوزه انجام شده می‌پردازیم. ادبیات و مفاهیم پایه را در بخش چهارم بیان می‌کنیم و در بخش پنجم، روش پیشنهادی ارائه می‌شود. در نهایت، نتایج را در بخش ششم بررسی و تحلیل می‌کنیم و در انتها در بخش‌های هفتم و هشتم به ترتیب، نتیجه‌گیری و مراجع آورده می‌شود.

۲- انگیزه پژوهش

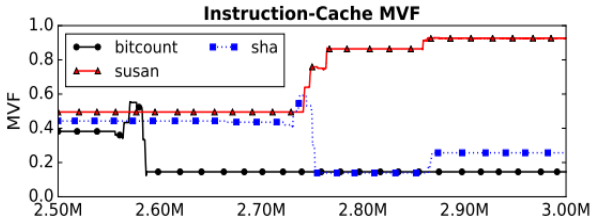
به دلیل تفاوت در برنامه‌های مختلف و نحوه دسترسی متفاوت آنها به حافظه، به ازای حجم یکسانی از حافظه نهان داده یا دستورات، زمان اجرای برنامه‌های مختلف نسبت به یکدیگر متفاوت است. همین دسترسی‌های متفاوت برنامه‌ها به حافظه باعث می‌شود آسیب‌پذیری حافظه نهان برنامه‌های مختلف متفاوت باشد. از طرفی، آسیب‌پذیری حافظه نهان با زمان اجرای برنامه‌ها رابطه مشخصی ندارد. این مسأله در شکل ۱ نشان داده شده است. بر طبق این نمودارها، آسیب‌پذیری برنامه‌ها با یک حافظه نهان یکسان با یکدیگر متفاوت است که با زمان اجرای برنامه‌ها نیز رابطه مستقیمی ندارد. بدین معنی که برنامه‌ای با زمان اجرای بالاتر، لزوماً آسیب‌پذیری بالاتری در مقایسه با برنامه‌ای با زمان اجرای کم‌تر ندارد (مقایسه آسیب‌پذیری حافظه نهان داده و زمان اجرای برنامه‌های patricia و dijkstra) و همچنین برنامه‌ای با زمان اجرای کم‌تر، آسیب‌پذیری بالاتری نسبت به برنامه‌ای با زمان اجرای بالاتر ندارد (مقایسه آسیب‌پذیری حافظه نهان داده و زمان اجرای برنامه‌های bitcount و susan). با این اوصاف لازم است اندازه حافظه نهان مناسبی متناسب با برنامه‌های کاربردی مختلف مشخص شود.



شکل ۱. زمان اجرا و آسیب‌پذیری حافظه نهان داده برنامه‌های محک مختلف در شبیه‌ساز gem5

از طرفی، آسیب‌پذیری داده‌های حافظه نهان به الگوی دسترسی‌ها به آدرس‌های مختلف آن بستگی دارد و از آن‌جایی که برنامه‌های مختلف الگوهای متفاوتی در دسترسی به حافظه دارند، تخمین و یا پیش‌بینی مقدار آسیب‌پذیری کار پیچیده و چالش‌برانگیزی است. در شکل ۲، آسیب‌پذیری چند برنامه در حین زمان اجرا نمایش داده شده است [۲۲]. این نمودار نشان می‌دهد که قابلیت اطمینان حافظه نهان در طول اجرای برنامه ثابت نیست. این تغییرات برای برنامه‌های مختلف نیز متفاوت است و هر برنامه می‌تواند در فازهای مختلفی از آسیب‌پذیری قرار گیرد. در نتیجه، علاوه بر اینکه اندازه حافظه نهان می‌تواند آسیب‌پذیری را تغییر دهد، آسیب‌پذیری در زمان اجرا نیز می‌تواند متفاوت باشد و مورد توجه قرار گیرد.

با توجه به تغییر آسیب‌پذیری حافظه نهان در طول اجرای برنامه‌ها و ارائه معیار اندازه موثر حافظه نهان برای برنامه‌های کاربردی مختلف [۲۱]، این ایده به ذهن رسید که از تغییر حجم حافظه نهان در جهت بهبود قابلیت اطمینان آن استفاده کنیم. با توجه به تفاوت رفتار آسیب‌پذیری حافظه در طول اجرای برنامه‌ها و چالش‌های مربوط به تخمین آسیب‌پذیری حافظه نهان در زمان اجرا، تا کنون روشی مبتنی بر تخمین میزان آسیب‌پذیری حافظه نهان در زمان اجرا و تغییر حجم حافظه نهان بر مبنای آن برای بهبود آسیب‌پذیری یا مدت اجرای برنامه ارائه نشده است.



شکل ۲. فازهای مختلف ضریب آسیب‌پذیری حافظه نهان دستورات در طول اجرا برای برنامه‌های مختلف [۲۲]

۳- کارهای پیشین

پیشرفت تکنولوژی باعث افزایش میزان حافظه نهان هر پردازنده و افزایش چگالی تعداد هسته‌های یک سیستم شده است. افزایش حافظه نهان، خطای نرم آن را به صورت ابرخطی[۱] افزایش می‌دهد [۲۰،۲۶،۲۷]. علاوه بر این، با افزایش اندازه حافظه نهان، سربار تکنیک‌های حفاظتی افزایش می‌یابد، زیرا برای حفظ نرخ خطای نرم در حد معقول باید به تعداد فزاینده‌ای از بلوک‌ها در مدت زمان ثابتی دسترسی داشت [۲۸،۲۹]. استفاده از کدهای تشخیص و تصحیح خطا، روش غالب محافظت در برابر خطای نرم در سلول‌های حافظه است [۳۰]. با این حال، روش‌های متعددی از روش‌های تکرار[۲] برای بالا بردن قابلیت اطمینان حافظه استفاده کرده‌اند و تعدادی نیز سعی در کاهش احتمال بروز خطای نرم در حافظه کرده‌اند.

در [۳۱] یک رویکرد حفاظت در برابر خطا برای حافظه نهان سطح ۱ پیشنهاد می‌شود که از محلیت مکانی[۳] حافظه نهان استفاده می‌کند. از آنجایی که در حافظه نهان، بخش کوچکی از داده‌ها متناوباً مورد دسترسی قرار می‌گیرند، در این تکنیک، از مدارهای جداگانه‌ای تنها برای محافظت از آن بخش استفاده می‌شود. در مقایسه با رویکرد مرسوم ارائه تصحیح خطا برای همه بلوک‌های حافظه نهان، این تکنیک سطح بالاتری از حفاظت را ارائه می‌کند. گروه Zhang، یک طرح تکرار درون حافظه‌ای برای محافظت از حافظه نهان پیشنهاد کرده است [۳۲]. در این تکنیک، بلوک‌هایی که برای مدت معینی مورد استفاده قرار نگرفته‌اند به عنوان بلوک‌های مرده تعیین می‌شوند و از فضای این بلوک‌ها برای نگهداری کپی بلوک‌های فعال حافظه نهان استفاده می‌شود. آنها دو استراتژی تکرار را بررسی کرده‌اند. در استراتژی اول، یک بلوک در موارد باخت[۴] و دسترسی نوشتن به حافظه نهان، تکرار می‌شود در حالی که در استراتژی دوم، یک بلوک تنها در دسترسی‌های نوشتن تکرار می‌شود. استراتژی دوم، داده‌های فقط خواندنی را تکرار نمی‌کند و از این رو، می‌تواند درصد بیشتری از داده‌های دستکاری

[۱] Superlinear
[۲] Redundancy
[۳] Temporal locality
[۴] Miss

شده را تکرار کند. به همین دلیل، تعادل خوبی از کارآیی و قابلیت اطمینان را فراهم می‌کند.

همین گروه در [۳۳] حافظه نهان تکراری (R-cache[۱]) را ارائه کرده‌اند که در آن از یک حافظه نهان تداعیگر[۲] کوچک اضافه استفاده شده است تا در هر بار نوشتن در حافظه نهان داده سطح ۱، یک کپی از داده‌ها را در آن نگهداری کنند. با توجه به محلیت ارجاع بالا در حافظه نهان سطح ۱، کپی تنها چند بلوک (مثلاً ۸ بلوک) می‌تواند برد دسترسی خواندن مورد دسترسی را فراهم کند. با این حال، استفاده از حافظه نهان تکراری برای حافظه نهان سطح ۲ و بالاتر، به دلیل کاهش محلیت مکانی و زمانی غیرممکن می‌شود [۳۴،۳۵]. به همین دلیل، یک حافظه نهان تکرار با اندازه بزرگ مورد نیاز است که به دلیل طراحی کاملاً تداعیگر آن، سربار بالای انرژی دینامیکی را تحمیل می‌کند.

در [۳۶] رویکردی برای اطمینان از قابلیت اطمینان حافظه نهان با حداقل سربار پیشنهاد شده است. در این طرح، تنها بلوک‌های کثیف حافظه نهان با استفاده از کدهای تصحیح خطا محافظت می‌شوند. در مقابل، بلوک‌های تمیز فقط با استفاده از کد توازن[۳] محافظت می‌شوند که نیازمندی‌های کدهای تصحیح خطا را کاهش می‌دهد. علاوه بر این، به طور دوره‌ای بلوک‌های کثیف حافظه نهان که انتظار نمی‌رود در آینده نزدیک دسترسی شوند، در حافظه نهان سطح بالاتر نوشته می‌شوند. این مسأله تعداد بلوک‌های کثیف حافظه نهان را کاهش می‌دهد و ترافیک حافظه اصلی را نیز به میزان قابل توجهی افزایش نمی‌دهد.

در [۳۷] رویکردی برای حفاظت در برابر خطا با استفاده از حافظه نهان محافظت شده جزئی ارائه شده است. در این روش چندین (مثلا دو) حافظه در یک سطح سلسله مراتب حافظه استفاده می‌شود و هر آدرس حافظه منحصراً به یکی از این حافظه‌ها نگاشت می‌شود. یکی از حافظه‌ها با استفاده از کدهای تشخیص و تصحیح خطا در برابر خطاهای نرم محافظت می‌شود و دیگری در برابر خطاهای نرم محافظت نمی‌شود. داده‌های برنامه به بخش‌های بحرانی و غیر بحرانی تقسیم می‌شود و اولی در حافظه نهان محافظت شده و دومی به حافظه نهان محافظت نشده نگاشت می‌شود. نشان داده شده است که با این روش، نرخ خرابی برنامه‌های چندرسانه‌ای را می‌توان همانند معماری با یک حافظه نهان محافظت شده حفظ کرد در حالی که هزینه‌های سربار را با حداقل کاهش کیفیت خدمات به حداقل رساند.

تعداد زیادی از بلوک‌های حافظه نهان، داده‌های «مرده» را ذخیره می‌کنند که در آینده مورد استفاده قرار نخواهند گرفت [۲۹،۳۵،۳۸]. این بلوک‌های داده در برابر خطاهای نرم آسیب‌پذیر باقی می‌مانند. برای رفع این مشکل، برخی از محققان تکنیک‌هایی را برای بازنویسی داده‌های کثیف یا خالی کردن حافظه نهان ارائه می‌کنند تا احتمال خراب شدن بلوک‌های آسیب‌پذیر در حافظه نهان را کاهش دهند. اسدی و همکاران، نوشتن داده‌های کثیف حافظه نهان سطح ۱ به حافظه نهان سطح ۲ به صورت دوره‌ای (مثلا هر یک میلیون سیکل) را پیشنهاد داده‌اند [۲۷]. لی و همکاران یک سیاست بازنویسی اولیه را پیشنهاد کرده‌اند که داده‌های کثیف حافظه نهان سطح ۱ را پس از گذشت یک دوره زمانی ثابت (مثلاً ۱۰ هزار چرخه) از آخرین عملیات نوشتن در یک بلوک، بازنویسی می‌کنند [۳۹]. گلد و همکاران یک مکانیزم پیش‌بینی ارائه کرده‌اند که به‌دقت تعیین می‌کند یک بلوک حافظه نهان برای آخرین بار چه زمانی نوشته می‌شود و در همان زمان بازنویسی داده‌ها را آغاز می‌کند [۴۰].

کادایف و همکاران تکنیک‌هایی را برای بهبود قابلیت اطمینان حافظه نهان ارائه کرده‌اند [۴۱]. اولین تکنیک آنها فقط کلمات اصلاح شده یک بلوک حافظه نهان را بازنویسی می‌کند تا از انتشار خطا به سطوح پایین‌تر سلسله مراتب حافظه جلوگیری کند. تکنیک دوم آن‌ها به‌طور انتخابی بلوک‌های حافظه نهان را نامعتبر می‌کند تا دوره‌های آسیب‌پذیر آن‌ها را کم کند و شانس آن‌ها را برای گرفتن خطاهای نرم کاهش دهد. برای به حداقل رساندن سربار کارآیی نامعتبر کردن بلوک‌های حافظه نهان، آنها تکنیک دیگری را پیشنهاد کرده‌اند که تلاش می‌کند نسخه جدیدی از بلوک نامعتبر را از طریق واکشی اولیه به حافظه نهان بیاورد. هرچند این تکنیک بخشی از افزایش قابلیت اطمینان حاصل از نامعتبر کردن بلوک‌های حافظه را کاهش می‌دهد.

در [۴۲] تکنیکی برای حافظه نهان سطح ۱ مستقیم نویسی[۴] که توسط کدهای تصحیح خطا محافظت نمی‌شود ارائه شده است. در این روش فرض شده است که حافظه نهان سطح ۲ و حافظه اصلی توسط کدهای تصحیح خطا محافظت می‌شوند. آنها نشان داده‌اند که افزایش استفاده از گذرگاه سطح ۲ به دلیل افزایش دسترسی به آن جهت کاهش آسیب‌پذیری حافظه سطح ۱ اندک است. تکنیک آنها به طور انتخابی بلوک‌های حافظه نهان را از حافظه نهان سطح ۲ به حافظه نهان سطح ۱ واکشی می‌کند تا بلوک‌های حافظه نهان را تازه

[۱] Replication cache
[۲] Fully associative
[۳] Parity
[۴] Write through

کند و آسیب‌پذیری حافظه نهان سطح ۱ را کاهش دهد. آنها دو استراتژی بازیابی را پیشنهاد کرده‌اند. استراتژی «استاتیک» بلوک‌های حافظه نهان انتخابی را با فرکانس ثابت بازیابی می‌کند. استراتژی «رویداد محور[۱]» یک واکشی مجدد را در یک برد در دسترسی خواندن سطح ۱ به بلوک‌هایی که انتظار می‌رود در آینده نزدیک به آن‌ها دسترسی پیدا کنند، راه‌اندازی می‌کند. در مقایسه با استراتژی اول، استراتژی دوم با هدف در نظر گرفتن الگوی دسترسی برای کاهش موثر آسیب‌پذیری انجام می‌شود.

۴- ادبیات و مفاهیم پایه‌ای

۴-۱- مدل قابلیت اطمینان حافظه نهان

قابلیت اطمینان رابطه معکوسی با آسیب‌پذیری حافظه نهان دارد. آسیب‌پذیری حافظه نهان به صورت احتمال بروز خطا در خروجی سیستم در صورت تغییر ناخواسته در هر یک از بیت‌های حافظه نهان تعریف می‌شود. برای تخمین قابلیت اطمینان حافظه نهان، نیاز به مدل جامعی از قابلیت اطمینان حافظه نهان است. برای این منظور، در این مقاله از مدل تحلیلی برای محاسبه قابلیت اطمینان استفاده شده است. قابلیت اطمینان یک ساختار در سیستم‌های دیجیتال با رابطه ۱ مدل می‌شود [۴۳]:

$$FIT_{structure} = FIT_{intrinsic} \times N \times AVF_{structure} \qquad (۱)$$

که $FIT_{intrinsic}$، ثابت وابسته به تکنولوژی است و تعداد خطا در واحد زمان آن را نشان می‌دهد. N اندازه ساختار را بر اساس تعداد بیت‌های آن نشان می‌دهد و AVF[۲] ضریب آسیب‌پذیری معماری است. این معیار، خطاهای قابل مشاهده از سوی کاربر را مشخص می‌کند و احتمال تأثیر خرابی بیت‌ها را در خروجی بیان می‌دارد [۴۳]. در مورد حافظه نهان، ضریب آسیب‌پذیری حافظه نهان CVF[۳] که در [۴۴] معرفی شده است، نشان دهنده AVF آن است. این معیار بیان کننده احتمال انتشار خطا از حافظه نهان به دیگر اجزاء سیستم از جمله پردازنده و دیگر سطح‌های حافظه است. CVF را به صورت رابطه ۲ می‌توان بیان کرد:

$$CVF = \frac{cache\ vulnerability}{execution\ time \times cache\ size} \qquad (۲)$$

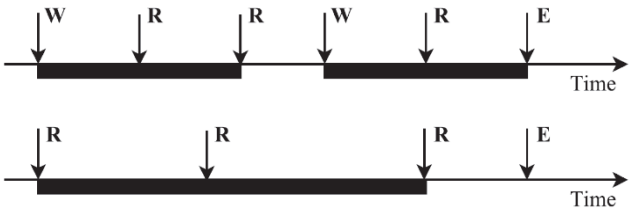
گروه Biswas در [۴۵] روش آنالیز طول عمر[۴] را برای ساختارهای بر پایه آدرس همانند حافظه نهان داده، ارائه کرده‌اند. در این روش، آسیب‌پذیری یک بیت در حافظه نهان به ترتیب دسترسی به آن بیت وابسته است. آسیب‌پذیری حافظه نهان، به صورت مجموع آسیب‌پذیری بیت‌های آن لحاظ می‌شود. در مورد حافظه نهان با مکانیزم پس‌نویسی، آسیب‌پذیری به صورت مجموع سیکل‌های آسیب‌پذیر که به خواندن یا خروج منتهی می‌شوند، تعریف می‌شود. شکل ۳ این مسأله را به صورت دقیق نشان می‌دهد. خطاهایی که در بازه‌های آسیب‌پذیر حافظه نهان اتفاق می‌افتد، احتمال دارد به دیگر اجزاء سیستم انتشار یافته و به صورت خطا خود را نشان دهد. بنابراین، با توجه به رابطه ۱ و رابطه ۲، مدل قابلیت اطمینان حافظه نهان را به صورت رابطه ۳ می‌توان بیان کرد:

$$FIT_{Cache} = FIT_{intrinsic} \times Cache\ Size \times CVF \qquad (۳)$$

با جایگزینی رابطه CVF در رابطه ۳، قابلیت اطمینان حافظه نهان به صورت رابطه ۴ بیان می‌شود:

$$FIT_{Cache} = \frac{FIT_{intrinsic} \times vulnerability}{execution\ time} \qquad (۴)$$

بر اساس رابطه ۴، قابلیت اطمینان حافظه نهان، رابطه مستقیمی با آسیب‌پذیری آن دارد. در این مقاله، منظور از قابلیت اطمینان حافظه نهان، FIT_{Cache} است. برای تخمین میزان آسیب‌پذیری، از روش ارائه شده در [۴۶] استفاده شده است.



شکل ۳. بازه‌های آسیب‌پذیر داده‌ها (بازه‌های تیره) در حافظه نهان [۲۵]

۴-۲- اندازه موثر حافظه نهان[۵]

برنامه‌های مختلف، نیازمندی‌ها و دسترسی‌های متفاوتی به حافظه دارند که همین موضوع باعث می‌شود رفتار آسیب‌پذیری متفاوتی در اندازه یکسان حافظه نهان نسبت به هم داشته باشند. از طرفی، افزایش و کاهش حافظه نهان برای هر برنامه‌کاربردی، آسیب‌پذیری حافظه نهان آن را در یکی از حالت‌های پایین، گذرا[۶] و بالا قرار می‌دهد [۲۱]. بنابراین، استفاده از حافظه نهان یکسان برای برنامه‌های مختلف، رفتار و فاز آسیب‌پذیری حافظه نهان آنها را یکسان نمی‌کند. در [۲۱] همچنین معیارهای مختلفی برای یکسان کردن رفتار آسیب‌پذیری حافظه نهان برنامه‌های کاربردی مورد بررسی قرار گرفته است. در نهایت، بر اساس تعداد دسترسی‌هایی که هر برنامه کاربردی به آدرس‌های مختلف دارد، معیاری به نام

[۱] Event-driven
[۲] Architectural Vulnerability Factor
[۳] Cache Vulnerability Factor
[۴] Lifetime Analysis
[۵] Effective Cache Size
[۶] Transient

اندازه موثر حافظه نهان معرفی شد. این معیار که ٪۹۵ بیشترین دسترسی‌ها در برنامه را شامل می‌شود، به راحتی و با اجرای یک بار برنامه کاربردی قابل محاسبه بوده و با افزایش و کاهش حافظه نهان بر اساس ضرایبی از آن باعث رفتار مشابه آسیب‌پذیری حافظه نهان برنامه‌ها می‌شود.

۵- ارائه روش پیشنهادی

۵-۱- تخمین آسیب‌پذیری حافظه نهان در زمان اجرا

محاسبه آسیب‌پذیری حافظه نهان، کاری پیچیده است و به راحتی از مشخصات سیستم همانند نرخ باخت، نرخ برد و ... قابل محاسبه نبوده و نیازمند اطلاعات دسترسی به حافظه نهان است [۲۲]. همان‌طور که بیان شد، آسیب‌پذیری حافظه نهان به صورت مجموع سیکل‌های آسیب‌پذیر بین دسترسی‌های مختلفی که به خواندن یا خروج منتهی می‌شوند، تعریف می‌شود. در این روش می‌بایست بازه‌های آسیب‌پذیر حافظه محاسبه و جمع شوند. بر این اساس، محاسبات آسیب‌پذیری حافظه نهان در زمان اجرا به صورت دقیق ممکن نیست و می‌بایست با روش‌هایی تخمین زده شود و بر اساس آن رفتار قابلیت اطمینان مورد بررسی قرار گیرد. در این مقاله، برای تخمین آسیب‌پذیری حافظه نهان از تعریف آن استفاده شده است. بر اساس تعریف، بازه‌های آسیب‌پذیر برای داده‌ها بر اساس نوع دسترسی‌ها به حافظه نهان داده مشخص می‌شود و عبارتند از:

۱. بازه‌هایی که منتهی به دسترسی خواندن هستند (که به عنوان بازه‌های access-read نام برده می‌شود)
۲. بازه‌های بین آخرین دسترسی یک داده کثیف تا خروج آن از حافظه نهان (که به عنوان بازه‌های dirty-evict شناخته می‌شوند).

در این راستا برای کاهش سربار پیاده‌سازی، تکنیک‌هایی رائه شده است که در ادامه تشریح می‌شوند.

۵-۱-۱- تخمین محاسبه بازه‌های dirty-evict

بازه‌های dirty-evict، بازه‌های زمانی آسیب‌پذیر بین آخرین دسترسی به یک آدرس کثیف تا خروج آن داده از حافظه نهان هستند. برای محاسبه چنین بازه‌هایی در سطح کلمات، نیاز به ثبت زمان آخرین دسترسی به ازائ هر کلمه کثیف می‌باشد تا در زمان خروج، زمان آسیب‌پذیری محاسبه شود. با توجه به سربار پیاده‌سازی چنین روالی، تخمینی معادل برای آن در نظر گرفته شده است. در این تخمین، زمان اولین دستور نوشتن[۱] در آن آدرس، ثبت شده و در زمان خروج، کل بازه اولین نوشتن تا خروج داده به عنوان بازه آسیب‌پذیر در نظر گرفته می‌شود. در چنین شرایطی، آسیب‌پذیری دسترسی‌های خواندن به یک آدرس کثیف در حافظه نهان لحاظ نمی‌شود، چرا که با بازه اولین نوشتن در آن آدرس تا خروج از حافظه نهان هم‌پوشانی دارد.

۵-۱-۲- تخمین محاسبات در سطح بلوک

با توجه به سربار ناشی از محاسبات آسیب‌پذیری در سطح کلمات، این محاسبات در سطح بلوک‌های حافظه نهان انجام شده است. هر دسترسی به یکی از آدرس‌های یک بلوک به منزله دسترسی به آن بلوک در نظر گرفته می‌شود. در این شرایط هر دسترسی خواندن به آدرسی در یک بلوک، مدت زمان بین دسترسی قبلی به این بلوک تا دسترسی فعلی به عنوان بازه آسیب‌پذیر در نظر گرفته می‌شود. همچنین، نوشتن در یک آدرس به معنای کثیف بودن کل داده‌ها است و بازه بین اولین دسترسی نوشتن به یکی از آدرس‌های آن بلوک تا خروج آن از حافظه به عنوان بازه آسیب‌پذیر در نظر گرفته می‌شود. در شرایط کثیف بودن داده‌ها، دسترسی‌های خواندن با dirty-evict تخمین‌زده شده هم‌پوشانی دارند. لذا فقط بازه dirty-evict در نظر گرفته می‌شود و دسترسی خواندن به داده کثیف، مورد محاسبه قرار نمی‌گیرد.

۵-۱-۳- تخمین در بازه‌های زمانی مشخص

برای تخمین آسیب‌پذیری در زمان اجرا، بازه‌های زمانی در نظر گرفته شده است که محاسبات مربوطه در این بازه‌ها انجام می‌شود. پیاده‌سازی این بازه‌های زمانی به وسیله شمارنده‌ای با تعداد بیت مشخص انجام شده که تعداد بیت شمارنده، مشخص کننده بازه زمانی تخمین آسیب‌پذیری حافظه است. مدت زمان این بازه‌ها باید به نحوی انتخاب شود که سربار محاسبات در آن دیده شود. بنابراین، این بازه‌ها نباید آن‌قدر بزرگ باشند که روند اصلی آسیب‌پذیری غیرقابل مشاهده باشد و نه آن‌قدر کوچک که محاسبات نوسانات آن زیاد و با اختلاف زیاد باشد.

۵-۲- تغییر حجم حافظه نهان در زمان اجرا

قابلیت اطمینان برنامه در طول زمان اجرا متغیر است. با به دست آمدن اندازه موثر حافظه نهان هر برنامه، می‌توان حجم مشخص شده از حافظه نهان بر مبنای این معیار در نظر گرفته شود. با به دست آمدن تخمینی از وضعیت آسیب‌پذیری در طول زمان اجرای برنامه، در صورتی که آسیب‌پذیری روندی رو به بالا داشته و برنامه در شرف ورود به فازی بود که آسیب‌پذیری آن بالا است، می‌توان با نصف

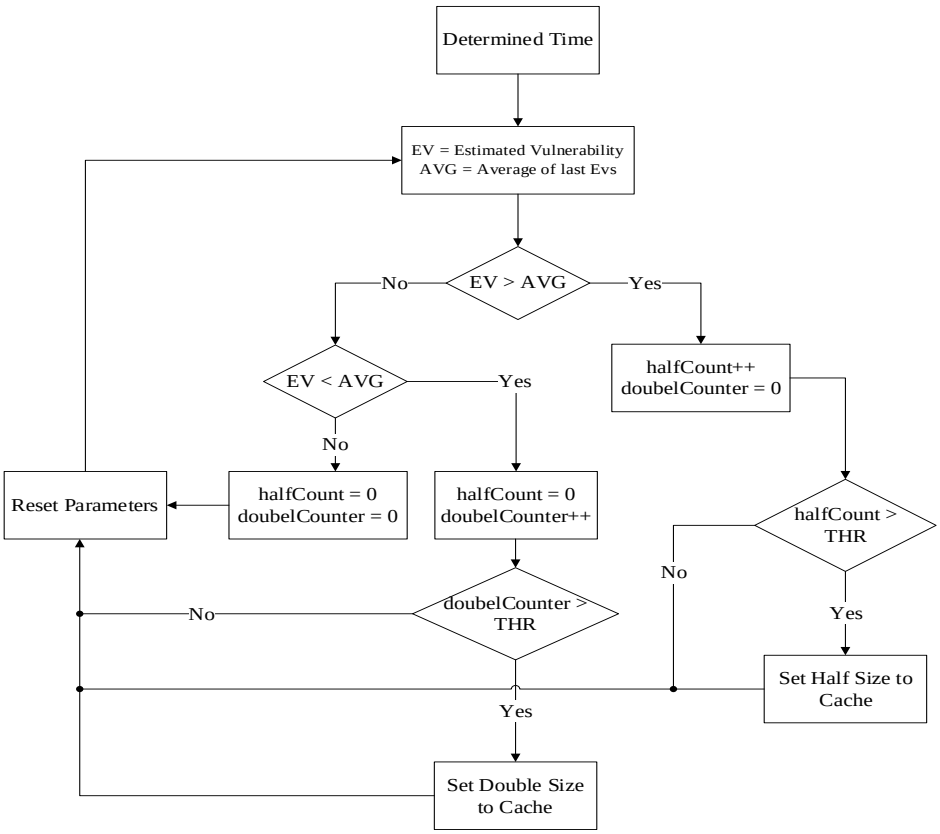
۱ Write

کردن حجم حافظه نهان، آسیب‌پذیری را کاهش داد تا برنامه در شرایط امن‌تری به کار خود ادامه دهد. در مقابل، در صورتی که روند آسیب‌پذیری رو به کاهش تشخیص داده شد، می‌توان حجم حافظه را دو برابر کرد تا برنامه که در فاز کم‌آسیب‌پذیر خود است، با سرعت بیشتری اجرا شود. برای نحوه بررسی روند آسیب‌پذیری بر مبنای تخمین به دست آمده در زمان اجرا، بازه زمانی ثابتی در نظر گرفته می‌شود. این بازه باید اولاً به اندازه‌ای بزرگ باشد که دسترسی‌های برنامه به حافظه نهان به تعدادی باشد که تخمین آسیب‌پذیری بر مبنای مدل ارائه شده قابل قبول باشد و ثانیاً، باید به حدی کوچک باشد که سربار پیاده‌سازی زیادی را تحمیل نکند. علاوه بر این، باید بتواند روند کاهشی یا افزایشی بودن آسیب‌پذیری را به خوبی مدل کند. در این بازه زمانی، تخمین آسیب‌پذیری بر مبنای مدل ارائه‌شده به دست می‌آید و با مقادیر آسیب‌پذیری تخمین زده شده در بازه‌های قبل مقایسه می‌شود تا کاهشی/افزایشی بودن روند مشخص شود. برای به دست آوردن بهتر روند آسیب‌پذیری، از روش پنجره لغزان و مقایسه آسیب‌پذیری با میانگین چند بازه قبل استفاده شده است تا نوسانات احتمالی بین بازه‌ها نیز به نوعی کاهش یابد. برای صدور دستور کاهش یا افزایش حجم حافظه نهان، باید چند کاهش یا افزایش متوالی صورت گیرد. لذا تعداد افزایش/کاهش‌های متوالی شمرده شده و در صورت بیشتر شدن آن از حدی مشخص، دستور مربوطه برای بازپیکربندی حافظه نهان صادر می‌شود. این روند در شکل ۴ به صورت فلوچارت نشان داده شده است.

۶- نتایج

۶-۱- شبیه‌سازی

در این کار از شبیه‌ساز Gem5 [۴۷] در حالت شبیه‌سازی SE[۱] برای اجرای برنامه‌ها استفاده شده است. سیستم پیکربندی شده دارای یک پردازنده تک هسته‌ای X86 با فرکانس ۱ گیگا هرتز است. این پردازنده دو حافظه نهان سطح اول، یکی برای دستورات و دیگری برای داده‌ها و یک حافظه اصلی دارد. با توجه به اینکه استخراج داده‌های تبادل شده در حافظه نهان مد نظر است، از مدل حافظه کلاسیک[۲] در حافظه نهان استفاده شده است. مکانیزم جایگزینی در حافظه نهان، LRU با روش پس‌نویسی[۳] و اندازه بلوک حافظه ۶۴ کلمه است.



شکل ۴. فلوچارت تغییر حجم حافظه نهان با توجه به شرایط آسیب‌پذیری

[۱] Syscall Emulator
[۲] Classic
[۳] Write Back

برای پردازنده‌های نهفته شده معمولاً از برنامه‌های محک[1] مجموعه MiBench استفاده می‌شود [۴۸]. در این مقاله نیز چند برنامه این مجموعه بر اساس جدول ۱ انتخاب شده است. این جدول دسته-بندی هر برنامه و توضیح کوتاهی در مورد هر یک را نشان می‌دهد. علاوه بر این، ردپای حافظه برنامه‌ها بر اساس فضای دسترسی آدرس‌ها در شبیه‌سازی‌ها ارائه شده است. همان‌طور که در جدول قابل مشاهده است، ردپای حافظه برنامه‌ها از 443.5×10^3 تا 8.4×10^6 متغیر است. برنامه‌های آزمایش شده دارای طیف گسترده‌ای از ردپای حافظه در برنامه‌های تعبیه شده هستند و همچنین بیشتر آنها در مطالعات مربوط به قابلیت اطمینان حافظه نهان در پردازنده‌های نهفته استفاده می‌شوند [۱۲،۱۷،۱۸]. در این مقاله، مطالعه بر روی داده‌های ورودی پیش فرض برنامه ها انجام انجام شده است. تمام آزمایش‌ها بر روی یک پردازنده Intel® Core™ i5-4460 با فرکانس ۳،۲ گیگاهرتز انجام شده است.

جدول ۱: برنامه‌های محک استفاده شده در شبیه‌سازی و اطلاعات مربوط به آنها

برنامه کاربردی	دسته	توضیحات	ردپای حافظه
basicmath	Auto./ industrial	عملیات ساده ریاضی	443.5 K
bitcount	Auto./ industrial	تست قابلیت دستکاری پردازنده با شمارش بیت‌های یک آرایه	825 K
CRC	Telecomm./ network	بررسی افزونگی چرخه ای ۳۲ بیتی بر روی فایل	4.3 M
dijkstra	Network	الگوریتم پیدا کردن کوتاه-ترین مسیر در ماتریس	941 K
FFT	Telecomm.	الگوریتم تبدیل فوریه سریع بر روی آرایه‌ای از داده‌ها	986.7 K
particia	Network	جستجوی درخت برای ارائه جدول مسیریابی شبکه	2 M
qsort	Auto./ industrial	الگوریتم جستجوی سریع	8.4 M
stringsearch	Office	جستجوی کلمه در ورودی	0.99 M
sha	Network/ security	الگوریم درهم‌آمیزی برای عملیات رمزنگاری	1.4 M
susan	Auto./ industrial	تشخیص گوشه/ لبه تصویر	1.4 M

۶-۲- نتایج بازه‌های زمانی تخمین آسیب‌پذیری

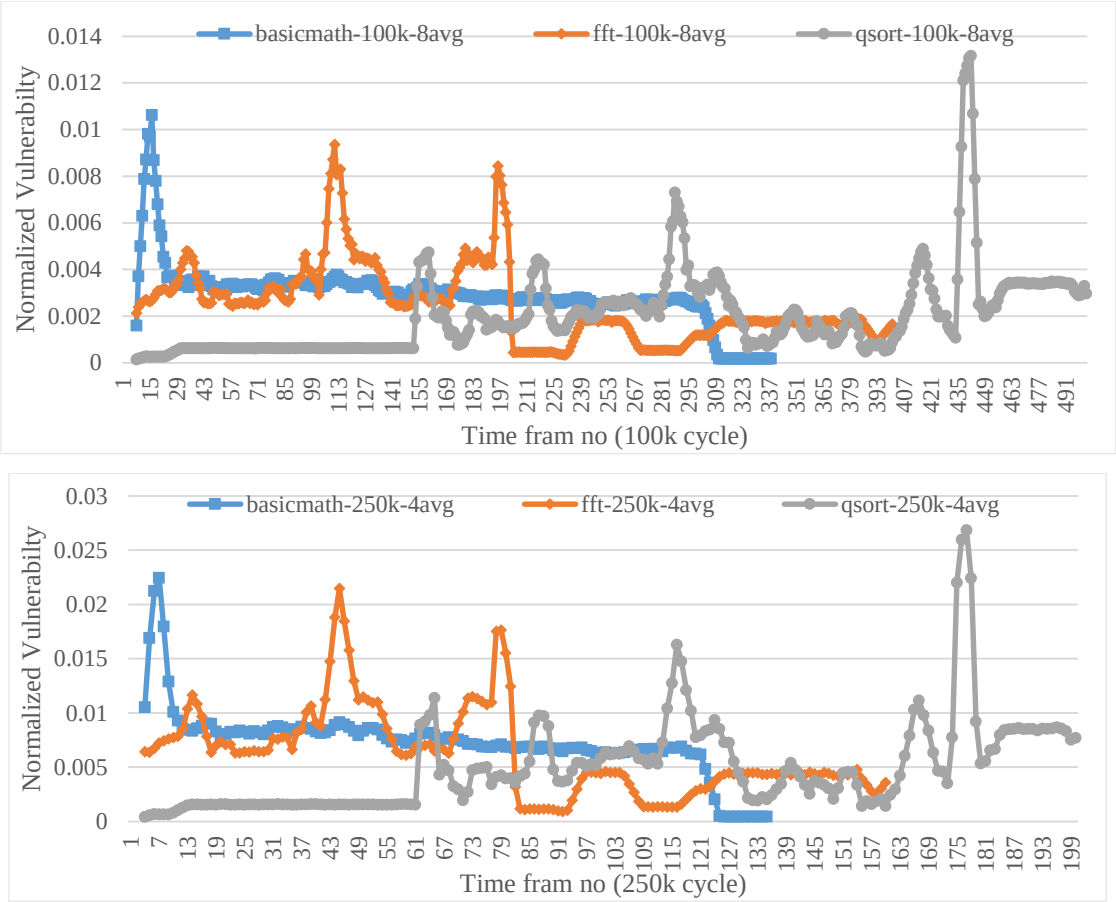
برای به دست آوردن بازه زمانی مناسب، یک بررسی روی تغییرات آسیب‌پذیری برنامه‌ها انجام شده است. در این بررسی که بر روی سه برنامه محک basicmath، FFT و qsort انجام شده، آسیب‌پذیری حافظه نهان در بازه‌های ده هزار سیکل (شمارنده ۱۴ بیتی)، پنجاه هزار سیکل (شمارنده ۱۶ بیتی)، صد هزار سیکل (شمارنده ۱۷ بیتی)، دویست و پنجاه هزار سیکل (شمارنده ۱۸ بیتی) و پانصد هزار سیکل (شمارنده ۱۹ بیتی) مورد بررسی قرار گرفته است.

با بررسی نمودارهای مربوطه، بازه‌های زمانی صد هزار سیکل و دویست و پنجاه هزار سیکل به دلیل نمایش روند آسیب‌پذیری و سربار محاسباتی مناسب، انتخاب شدند. برای دخیل کردن سابقه آسیب‌پذیری در تصمیم‌گیری‌ها، از مکانیزم میانگین‌گیری پنجره لغزان به اندازه ۸ و ۴ به ترتیب برای بازه‌های صد هزار سیکل و دویست و پنجاه هزار سیکل استفاده شده است. شکل ۵ نتیجه را برای کل زمان اجرای سه برنامه محک نشان می‌دهد. بر اساس این شکل‌ها، در هر دو حالت، هر نقطه روند آسیب‌پذیری برنامه‌ها را به صورتی نمایش می‌دهد که روند آسیب‌پذیری حافظه به خوبی قابل تشخیص است. حال این‌که کدام یک از این دو بازه مناسب برای طراحی نهایی خواهند بود، باید از نظر دقت و سربارهای زمانی و مساحت، مورد بررسی قرار گیرند.

۶-۳- نتایج تخمین آسیب‌پذیری حافظه نهان در زمان اجرا

بر اساس بازه‌های زمانی در نظر گرفته شده برای محاسبه آسیب‌پذیری در زمان اجرا (صد هزار و دویست و پنجاه هزار سیکل)، یک شبیه‌سازی برای محاسبه دقت تخمین آسیب‌پذیری در زمان اجرا انجام شده است. براساس الگوریتم گفته شده، یک جدول برای ثبت زمان دسترسی‌ها بر مبنای یک شمارنده ۱۶ بیتی در نظر گرفته شده است. هر سطر این جدول مربوط به یک بلوک از حافظه نهان است که در آن مقدار شمارنده بازه زمانی (به عنوان زمان دسترسی به بلوک متناظر با آن سطر از حافظه نهان) ثبت می‌شود. با دسترسی بعدی به هر بلوک، متناسب با نوع دسترسی و روش گفته شده، میزان آسیب‌پذیری به‌روز می‌شود. در انتهای یک دوره کامل از بازه زمانی، برای تمامی بلاک‌های کثیف، بازه دسترسی ثبت شده در جدول تا پایان بازه، به عنوان بازه‌های آسیب‌پذیر به مجموع آسیب‌پذیری این بازه اضافه می‌شود.

[1] Benchmark



شکل ۵. تغییرات آسیب‌پذیری حافظه نهان سه برنامه در زمان اجرا با بازه‌های زمانی صد هزار سیکل و میانگین‌گیری هشت تایی و دویست و پنجاه هزار سیکل و میانگین‌گیری چهار تایی در هر نقطه

برای بررسی دقت محاسبات در شبیه‌سازی انجام شده، حجم حافظه نهان هر برنامه بر اساس معیار اندازه موثر آن برنامه در نظر گرفته شده است و پیکربندی آن نگاشت مستقیم است.

جدول ۲جدول ۲ نتایج به دست آمده از شبیه‌سازی محاسبات بلوکی نسبت به حالت ایده‌آل را برای بازه‌های زمانی صد هزار و دویست و پنجاه هزار سیکل نشان می‌دهد. بر اساس نتایج این جدول، دقت محاسبات بلوکی در بازه‌های صد هزار سیکل در بدترین حالت ۵۰٪ و در بهترین حالت ۶۵٪ است. در بازه‌های دویست و پنجاه هزار سیکل نیز دقت تخمین آسیب‌پذیری بین ۵۳٪ تا ۷۳٪ است. بر اساس این نتایج مشاهده می‌شود که دقت محاسبات در بازه‌های دویست و پنجاه هزار سیکل به صورت کلی بهتر است. کاهش دقت محاسبات ناشی از انجام محاسبات در سطح بلوک به جای کلمه و محدود کردن سطرهای جدول به ۱۶ بیت برای کاهش سربار پیاده-سازی سخت‌افزاری می‌باشد.

جدول ۲: نتایج دقت محاسبات تخمین آسیب‌پذیری بیتی

program	100k cycles estimation accuracy	250k cycles estimation accuracy
basicmath	63%	73%
bitcount	50%	55%
crc	61%	55%
dijkstra	50%	52%
fft	61%	54%
patricia	60%	53%
qsort	50%	53%
rijenc	62%	64%
strsearch	59%	55%
sha	65%	58%
susan	50%	56%

۶-۴- نتایج تخمین روند آسیب‌پذیری حافظه نهان در زمان اجرا

پس از بررسی تخمین آسیب‌پذیری حافظه نهان، الگوریتم ارائه شده برای تشخیص روند آسیب‌پذیری مورد بررسی قرار گرفت. همان‌طور که بیان شد، هدف در این کار، تشخیص درست روند آسیب‌پذیری حافظه نهان برای تغییر اندازه حافظه نهان با حداقل سربار پیاده-سازی است. بر اساس الگوریتم ارائه شده، روند آسیب‌پذیری برای

دو بازه زمانی صد هزار و دویست و پنجاه هزار سیکل مورد بررسی قرار گرفت. روش بررسی به این شکل است که در انتهای هر بازه زمانی، مقدار تخمین زده شده برای آسیب‌پذیری با میانگین مقادیر بازه‌های قبل با تعداد مشخص، مقایسه می‌شود. برای تشخیص روند در بازه‌های صد هزار سیکل، مقدار محاسبه شده آسیب‌پذیری با ضریب ۰/۷۵، با میانگین ۸ بازه قبل مقایسه می‌شود. در بازه‌های با طول دویست و پنجاه هزار سیکل، مقدار تخمینی با میانگین ۴ تخمین قبل مقایسه می‌گردد.

جدول ۳ نتیجه شبیه‌سازی‌ها را نشان می‌دهد. بر اساس نتایج جدول، دقت تشخیص روند در بازه دویست و پنجاه هزار سیکل بین ۸۷/۹٪ تا ۹۹/۳۸٪ با میانگین ۹۵/۲۲٪ و دقت تصمیم‌گیری برای بازه‌های صد هزار سیکل بین ۷۵/۲۵٪ تا ۹۹/۰۲٪ با میانگین ۹۰/۲۴٪ است. همان‌طور که ملاحظه می‌شود، تشخیص روند آسیب‌پذیری حافظه نهان برای هر دو بازه قابل قبول است ولی از آنجایی که نتایج برای بازه‌های دویست و پنجاه هزار سیکلی بهتر است، برای پیاده‌سازی، بازه‌های دویست و پنجاه هزار سیکلی در نظر گرفته شده است.

۶-۵- پیاده‌سازی ماژول تخمین و مدیریت آسیب‌پذیری حافظه نهان در زمان اجرا

الگوریتم تخمین و مدیریت آسیب‌پذیری در زمان اجرای ارائه شده با در نظر گیری بازه‌های زمانی دویست و پنجاه هزار سیکل، در VHDL پیاده‌سازی شده و سربارهای مساحت آن با ابزار Design Compiler به دست آمده است. همچنین جدول مورد نیاز این الگوریتم به صورت حافظه در نظر گرفته شده که مساحت آن از طریق شبیه‌ساز CACTI 6.5 به دست آمده است. در این پیاده‌سازی، از رجیسترهای ۳۲ بیتی برای نگهداری آسیب‌پذیری و رجیسترهای ۱۶ بیتی برای نگهداری زمان استفاده شده است.

جدول ۳: نتایج دقت تصمیم‌گیری الگوریتم تخمین و مدیریت آسیب‌پذیری در زمان اجرا

program	100k cycles decision accuracy	250k cycles decision accuracy
basicmath	97.33%	97.76%
bitcount	97.83%	98.04%
crc	99.02%	99.38%
dijkstra	93.56%	97.735%
fft	90.56%	91.75%
patricia	82.43%	94.5%
qsort	75.25%	87.9%
rijenc	91.9%	92.47%
strsearch	74.06%	89.1%
sha	95.13%	99.5%
susan	95.54%	99.3%
Average	**90.24%**	**95.22%**

مطابق نتایج به دست آمده، سربار مساحت پیاده‌سازی ماژول تخمین و مدیریت آسیب‌پذیری در زمان اجرا، برای حافظه نهان با حجم ۱ کیلوبایت (کوچک‌ترین حافظه مورد نیاز برنامه‌های این مقاله) ٪۵۴ و برای حافظه نهان با حجم ۶۴ کیلوبایت برابر ٪۵/۴ است. کم‌ترین طول سیکل اجرا برای این پیاده‌سازی برابر ۱/۱۴ نانو ثانیه به دست آمده است.

۶-۶- نتایج شبیه‌سازی مدیریت آسیب‌پذیری در زمان اجرا

برای بررسی اثر تغییر حجم حافظه نهان داده بر کارآیی اجرای برنامه‌ها و آسیب‌پذیری بر اساس الگوریتم ارائه شده، یک شبیه‌ساز حافظه نهان به زبان پایتون پیاده‌سازی شده است. ورودی این شبیه‌ساز، آدرس‌های مورد دسترسی برنامه‌ها است که از شبیه‌ساز Gem5 به دست آمده است. این شبیه‌ساز می‌تواند در زمان اجرا، بر اساس الگوریتم ارائه شده، برای تخمین و مدیریت آسیب‌پذیری در زمان اجرا، حجم حافظه نهان را کم یا زیاد کند و سربارهای زمانی را با توجه به پیاده‌سازی انجام شده در بخش قبل لحاظ کند. برای اجرای هر برنامه در این شبیه‌ساز فرض شده است که هر برنامه حجمی مناسب از نظر برقراری مصالحه قابلیت اطمینان و کارآیی بر مبنای معیار اندازه موثر حافظه نهان دارد. بنابراین، در ابتدای شروع اجرای هر برنامه، حجم حافظه نهان داده بر اساس اندازه موثر هر برنامه انتخاب شده است و حجم حافظه نهان فقط می‌تواند نصف یا دو برابر مقدار اولیه شود.

در شبیه‌سازی تغییر حجم حافظه نهان در زمان اجرا، چند مسئله در رابطه با سربار اجرایی برنامه‌ها مطرح می‌شود. در انتهای هر بازه زمانی، سربار محاسبات ناشی از داده‌های کثیف برای تخمین آسیب‌پذیری وجود دارد. در این مورد، در بدترین حالت، حجم جدول برای بزرگ‌ترین حافظه برابر ۱۰۲۴ خانه است که اگر داده همه خانه‌ها کثیف باشد، سربار زمانی محاسبات آن حدود ٪۰/۴ است.

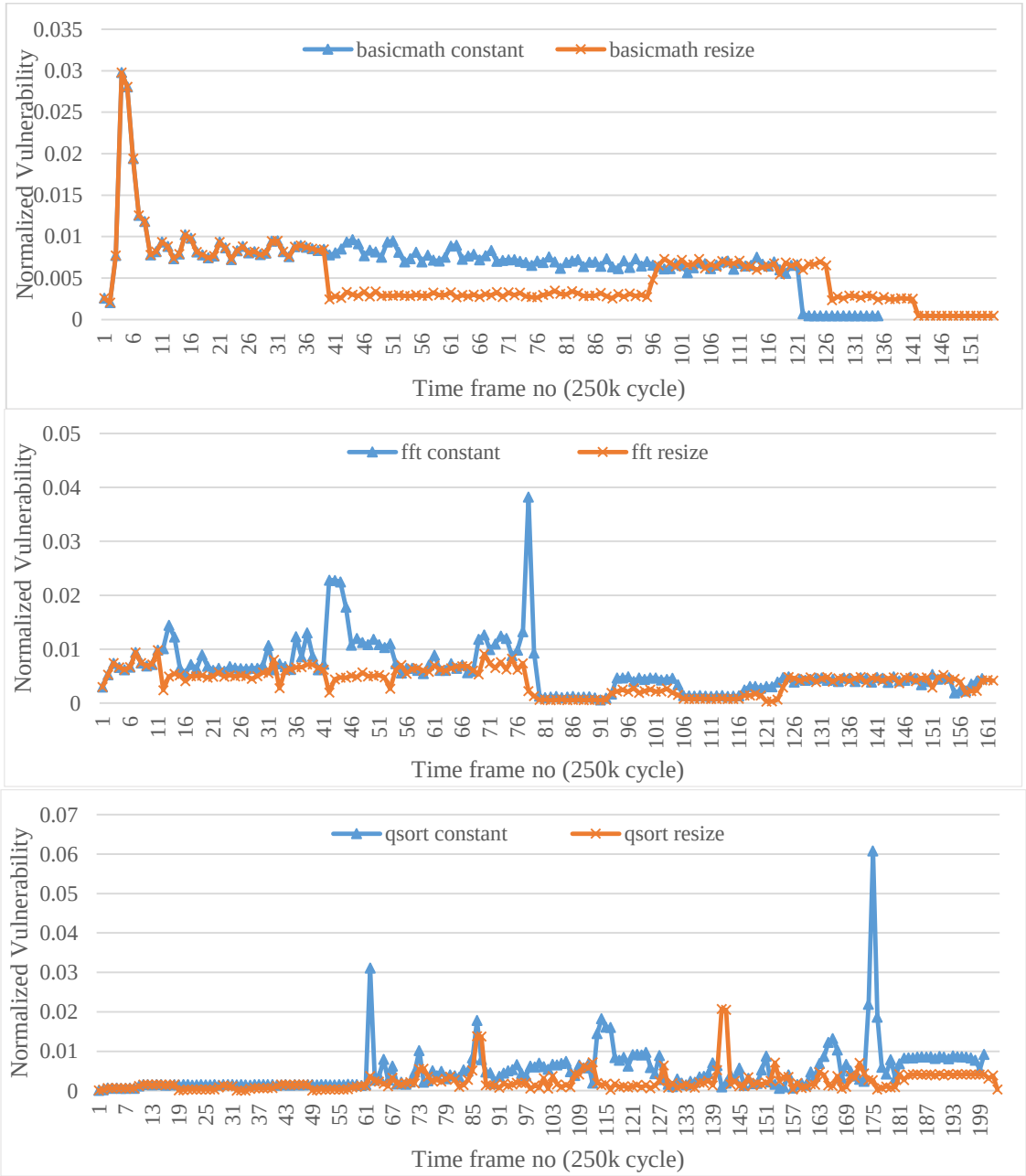
سربار دیگر می‌تواند ناشی از صدور دستور تغییر حجم حافظه نهان باشد. در این شرایط، داده‌های کثیف باید در حافظه پس‌نویسی شوند. پس از تغییر حجم حافظه نهان، برنامه از آخرین داده قبلی ادامه می‌یابد، اما حافظه نهان جدید خالی است که به مرور و با دسترسی‌های بعد پر می‌شود. همچنین بازپیکربندی حافظه نهان و کنترلر آن نیز ممکن است چند سیکل اضافه به سیستم تحمیل کند. این موضوع بر اساس مرجع [۴۹]، ۱۰۰ سیکل در نظر گرفته شده است. لذا به صورت خلاصه سربارهای زمانی ماژول مدیریت آسیب‌پذیری در زمان اجرا عبارتند از:

۱. سربار ناشی از تخمین آسیب‌پذیری در انتهای هر بازه ۲۵۰ هزار سیکل (حدود ۰/۴ درصد)

۲. سربار ناشی از پس‌نویسی داده‌های کثیف پیش از تغییر حجم حافظه
۳. سربار ناشی از بازپیکربندی حجم حافظه نهان در زمان اجرای برنامه که ۱۰۰ سیکل است.
۴. سربار ناشی از ادامه برنامه با حافظه نهان جدید.

هر کدام از این سربارها در شبیه‌ساز لحاظ شده است. جدول ۴ نتایج شبیه‌سازی برنامه‌های محک را نشان می‌دهد. همان‌طور که مشاهده می‌شود، زمان اجرای برنامه‌ها به صورت میانگین ٪۶ طولانی‌تر می‌شود اما در مقابل، آسیب‌پذیری کل ٪۳۶ کاهش می‌یابد.

در شکل ۶ مقایسه تغییرات آسیب‌پذیری در زمان اجرا برای سه برنامه basicmath، FFT و qsort در دو حالت اجرا با حافظه نهان داده ثابت و اجرا با الگوریتم تخمین و مدیریت آسیب‌پذیری رسم شده است. همان‌طور که ملاحظه می‌شود هرگاه آسیب‌پذیری در حجم ثابت (خط آبی با علامت مثلث) روندی رو به افزایش یا کاهش داشته است، حجم حافظه تغییر یافته و سطح آسیب‌پذیری برنامه در اجرا با الگوریتم مدیریت آسیب‌پذیری (خط قرمز با علامت ضربدر) کم یا زیاد شده است. همچنین در تمام طول اجرا، آسیب‌پذیری حافظه نهان در اجرا با الگوریتم مدیریت آسیب‌پذیری پایین‌تر از آسیب‌پذیری در اجرا با حجم ثابت بوده است.



شکل ۶. مقایسه آسیب‌پذیری برنامه‌های basicmath، FFT و qsort در زمان اجرا، در اجرا با حجم حافظه نهان داده ثابت و اجرا با الگوریتم تخمین و مدیریت آسیب‌پذیری در زمان اجرا

جدول ۴: نتایج به دست آمده از شبیه‌ساز تغییر حجم حافظه نهان بر اساس مدیریت آسیب‌پذیری در زمان اجرا

program	Execution time increase	Vulnerability decrease
basicmath	1.14	0.79
bitcount	1.02	0.61
crc	1.06	0.85
dijkstra	1.03	0.83
fft	1.01	0.72
patricia	1.07	0.72
qsort	1.02	0.53
rijenc	1.01	0.90
strsearch	1.08	0.58
sha	1.05	0.87
susan	1.15	0.77
average	1.06	0.74

۷- نتیجه‌گیری

در این مقاله، الگوریتمی برای تخمین و مدیریت آسیب‌پذیری حافظه نهان در زمان اجرا ارائه شد. مطابق این الگوریتم ماژولی در کنار حافظه نهان قرار می‌گیرد که در بازه‌های زمانی مشخص، تخمینی از آسیب‌پذیری آن ارائه می‌کند و با رصد کردن روند تغییرات آسیب‌پذیری در زمان اجرا، دستوری برای بازپیکربندی حافظه نهان می‌دهد. این ماژول در صورت تشخیص افزایش آسیب‌پذیری، حجم حافظه نهان را کاهش و در صورت کم شدن آسیب‌پذیری، دستور به افزایش حجم آن می‌دهد. دقت تصمیم‌گیری این الگوریتم نسبت به حالت ایده‌آل بین ٪۸۷/۹ تا ٪۹۹/۳۸ است.

بر اساس پیاده‌سازی صورت گرفته از الگوریتم تخمین و مدیریت آسیب‌پذیری حافظه نهان، سربار مساحت پیاده‌سازی این ماژول برای بزرگ‌ترین حافظه نهان ٪۵/۴ است. مطابق نتایج شبیه‌سازی، این ماژول تنها با ٪۶ افزایش میانگین زمان اجرای برنامه‌ها می‌تواند میانگین آسیب‌پذیری کل برنامه‌ها را ٪۳۶ بهبود بخشد و سطح آسیب‌پذیری حافظه نهان را در تمام طول اجرای برنامه از سطح آسیب‌پذیری در حجم حافظه ثابت پایین‌تر نگه دارد.

مراجع

[1] Shekhar Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degra dation," *IEEE Micro*, vol. 25, no. 6, pp. 10-16, Nov. 2005.

[2] B. W. Johnson, "Fault-Tolerant Microprocessor-Based Systems," *IEEE Micro*, vol. 4, no. 6, pp. 6-21, Dec. 1984.

[3] C. Slayman, "Soft error trends and mitigation techniques in memory devices," *in Proceedings of Annual Reliability and Maintainability Symposium*, Lake Buena Vista, FL, USA, 24-27 Jan. 2011.

[4] Isreal Koren and C. Mani Krishna, Fault-Tolerant Systems, Elsevier, 2007.

[5] Michael Nicolaidis, Soft Errors in Modern Electronic Systems, Springer, 2010.

[6] Lukasz G Szafaryn, Brett H Meyer, and Kevin Skadron. Evaluating overheads of multibit soft-error protection in the processor core. IEEE Micro, (4):56–65, 2013.

[7] Adam Neale, Maarten Jonkman, Manoj Sachdev, "Adjacent-MBU Tolerant SEC-DED-TAEC-yAED Codes for Embedded SRAMs," *IEEE Transactions on Circuit and Systems II*, vol. 62, no. 4, pp. 387-391, Apr. 2015.

[8] Seungyeob Lee, Joon-Sung Yang, "MVP ECC: Manufacturing process Variation aware unequal Protection ECC for memory reliability," *in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Switzerland, Mar. 27-31 2017.

[9] A. klockMann, G. Geogakos, M. Goessel, "A new 3-bit Burst-Error Correcting Code," *in Proceedings of IOLTS*, Greece, 3-5 Jul. 2017.

[10] R. Afrin and M. S. Sadi, "An efficient approach to enhance memory reliability," *in Proceedings of the 4th International Conference on Advances in Electrical Engineering (ICAEE)*, Dhaka, Bangladesh, 28-30 Sep. 2017.

[11] I. Alam, C. Schoeny, L. Dolecek and P. Gupta, "Parity++: Lightweight Error Correction for Last Level Caches," *in Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Luxembourg, 25-28 Jun. 2018.

[12] Alaa R. Alameldeen, Ilya Wagner, Zeshan Chishti, Wei Wu, Chris Wilkerson, Shih-Lien Lu, "Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes", *in Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, CA, USA, 4-8 Jun. 2011.

[13] JeongKyu Hong, Soontae Kim, "Smart ECC Allocation Cache Utilizing Cache Data Space," *IEEE Transaction on Computers*, vol. 66, issue 2, pp. 368-374, Feb. 2017.

[14] Henry Duve, Xun Jian, Rakesh Kumar, "Correction Prediction: Reducing Error Correction Latency for On-chip Memories," *in Proceedings of 21st International International Symposium on High Performance Computer Architecture (HPCA)*, CA, USA, 7-11 Feb. 2015.

[15] P. Benedicte, C. Hernandez, J. Abella and F. J. Cazorla, "LAEC: Look-Ahead Error Correction Codes in Embedded Processors L1 Data Cache," *in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019.

[16] Luc Jaulmes, Miquel MoretÃ³, Mateo Valero, Mattan Erez and Marc Casas. 'Runtime-Guided ECC Protection Using Online Estimation of Memory Vulnerability'. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '20 (Atlanta, GA, US, Nov. 2020).

[17] Wei Zhang, "Computing cache vulnerability to transient errors and its implication," in Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DTF'05), USA, 3-5 October 2005.

[18] Kooli, M., Di Natale, G. & Bosio, A. "Memory-Aware Design Space Exploration for Reliability Evaluation in Computing Systems," *Journal of Electronic Testing 35*, pp. 145–162, 2019.

[19] Jun Yan and Wei Zhang. "Evaluating instruction cache vulnerability to transient errors," *In Proceedings of the 2006 workshop on MEmory performance: DEaling with Applications, systems and architectures (MEDEA '06)*, New York, USA, Sep. 2006.

[20] Yuan Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi and S. M. Reddy, "Cache size selection for performance, energy and reliability of time-constrained systems," *in Proceesings of Asia and South Pacific Conference on Design Automation*, Yokohama, Japan, 24-27 Jan. 2006.

[21] M.H. Ahmadilivani, M. M. Jahromi, M.E. Salehi, M. Kargar, "ECS an endeavor towards providing similar cache reliability behavior in

[36] S. Kim, "Area-efficient error protection for caches," in Design, automation and test in Europe, 2006, pp. 1282–1287.

[37] K. Lee et al., "Mitigating soft error failures for multimedia applications by selective data protection," in international conference on Compilers, architecture and synthesis for embedded systems, 2006, pp. 411–420.

[38] S. Kaxiras et al., "Cache decay: exploiting generational behavior to reduce cache leakage power," in International symposium on Computer architecture (ISCA), 2001, pp. 240–251.

[39] L. Li et al., "Soft error and energy consumption interactions: a data cache perspective," in International Symposium on Low Power Electronics and Design (ISLPED), 2004, pp. 132–137.

[40] B. T. Gold et al., "Mitigating multi-bit soft errors in L1 caches using last-store prediction," in Proceedings of the Workshop on Architectural Support for Gigascale Integration, 2007.

[41] I. Kadayif et al., "Modeling and improving data cache reliability," in ACM SIGMETRICS Performance Evaluation Review, vol. 35, no. 1, 2007.

[42] V. Sridharan et al., "Reducing data cache susceptibility to soft errors," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 4, pp. 353–364, 2006.

[43] Shubu Mukhrejee, Architecture design for soft errors, Elsevier, 2008.

[44] S. Mukhejee, J. Emer, S.K. Reinhardt, The soft error problem: an architectural perspective, in: Proceedings of 11th International Symposium on High-performance Architecture, USA, 12–16 Feb, 2005, https://doi.org/10.1109/ HPCA.2005.37.

[45] A. Biswas *et al.*, "Computing architectural vulnerability factors for address-based structures," in *International Symposium on Computer Architecture (ISCA)*, 2005, pp. 532–543.

[46] M.H. Ahmadilivani, M.E. Salehi, M. Kargar, Effect of cache run-time parameters on the reliability of embedded systems, in: 2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST), Tehran, Iran, 10–11 Jun, 2020.

[47] Gem5 Simulator (2019). Gem5 home page [online]. Available: https://www.gem5. org/.

[48] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. MiBench: A free, commercially representative embedded benchmark suite. In International Workshop on Workload Characterization, pages 3–14, 2001.

[49] F. Kriebel, A. Subramaniyan, S. Rehman, S. J. B. Ahandagbe, M. Shafique and J. Henkel, "R2Cache: Reliability-aware reconfigurable last-level cache architecture for multi-cores," *in proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Amsterdam, Netherlands, 4-9 Oct. 2015.

different programs", *Microelectronics Reliability*, Volume 152, January 2024.

[22] A. Vijayan, S. Kiamehr, M. Ebrahimi, K. Chakrabarty and M. B. Tahoori, "Online Soft-Error Vulnerability Estimation for Memory Arrays and Logic Cores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 499-511, Feb. 2018.

[23] WeixunWang and Prabhat Mishra. 2011. Dynamic reconfiguration of two-level cache hierarchy in real-time embedded systems. J. Low Power Electron. 7, 1 (2011), 17–28.

[24] Weixun Wang, Prabhat Mishra, and Ann Gordon-Ross. 2012. Dynamic cache reconfiguration for soft real-time systems. ACM Trans. Embedded Comput. Syst. 11, 2 (2012), 28.

[25] Alif Ahmed, Yuanwen Huang, and Prabhat Mishra. 2019. Cache reconfiguration using machine learning for vulnerability-aware energy optimization. ACM Transactions on Embedded Computing Systems (TECS) 18, 2 (2019), 15.

[26] A. Biswas et al., "Explaining cache SER anomaly using DUE AVF measurement," in International Symposium on High Performance Computer Architecture (HPCA), 2010, pp. 1–12.

[27] G.-H. Asadi et al., "Balancing performance and reliability in the memory hierarchy," in International Symposium on Performance Analysis of Systems and Software (ISPASS), 2005, pp. 269–279.

[28] S. S. Mukherjee et al., "Cache scrubbing in microprocessors: Myth or necessity?" in IEEE Pacific Rim International Symposium on Dependable Computing, 2004, pp. 37–42.

[29] S. Mittal et al., "Improving energy efficiency of Embedded DRAM Caches for High-end Computing Systems," in 23rd International ACM Symposium on High Performance Parallel and Distributing Computing (HPDC), 2014, pp. 99–110.

[30] S. Mittal and J. S. Vetter, ''A survey of techniques for modeling and improving reliability of computing systems,'' IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 4, pp. 1226–1238, Apr. 2016.

[31] S. Kim et al., "Area efficient architectures for information integrity in cache memories," ACM SIGARCH Computer Architecture News, vol. 27, no. 2, pp. 246–255, 1999.

[32] W. Zhang et al., "ICR: In-Cache Replication for Enhancing Data Cache Reliability," in DSN, 2003, pp. 291–300.

[33] W. Zhang, "Replication cache: a small fully associative cache to improve data cache reliability," IEEE Transactions on Computers, vol. 54, no. 12, pp. 1547–1555, 2005.

[34] S. Mittal et al., "MASTER: A Multicore Cache Energy Saving Technique using Dynamic Cache Reconfiguration," IEEE Transactions on VLSI Systems, 2014.

[35] S. Mittal et al., "A Survey of Architectural Approaches for Managing Embedded DRAM and Non-volatile On-chip Caches," IEEE Transactions on Parallel and Distributed Systems (TPDS), 2014.