شمارنده‌های کارایی سخت‌افزاری

(قسمت اول)

فریبا غفاری
fariba.ghaffari@modares.ac.ir

منصوره قنادی
ma.ghanadi2006@gmail.com

آزمایشگاه تخصصی آپا در زمینه امنیت فن‌آوری اطلاعات و ارتباطات

http://cert.um.ac.ir
cert@um.ac.ir

ویرایش اول – فروردین‌ماه ۱۳۹۳

شماره سند: APA_FUM_W_MAL_0116

چکیده

در اغلب پردازنده‌های پیشرفته امروزی تراشه‌هایی وجود دارند که بر روی کارایی پردازنده نظارت می‌کنند. داده‌هایی که توسط این تراشه‌های سخت‌افزاری جمع‌آوری می‌شوند، اطلاعات مربوط به کارایی برنامه‌ها، سیستم‌عامل و پردازنده را ارائه می‌کنند. شمارنده‌های کارایی سخت‌افزاری مجموعه‌ای از ثبات‌های خاص منظوره درون این پردازنده‌ها هستند که فعالیت‌های مربوط به سخت‌افزار را شمارش کرده و در خود ذخیره می‌کنند. این اطلاعات به بهبود کارایی برنامه‌ها کمک بسیاری می‌کند.

واژه‌های کلیدی

کارایی، پردازنده، شمارنده کارایی سخت‌افزاری.

۱- مقدمه

مهندسین نرم‌افزار و برنامه‌نویسان علاوه بر تلاش برای اجرایی شدن کد، سعی می‌کنند در برخی از موارد کارایی برنامه خود را نیز افزایش دهند، دلیل آن هم بسیار ساده است. افزایش کارایی مزیت‌هایی گوناگونی دارد. برای مثال، مطالعه کد و کنترل کارایی آن به برنامه‌نویس این امکان را می‌دهد تا در محیطی کوچک و دقیق‌تر برنامه خود را اجرا و شبیه‌سازی کرده و به تبع آن می‌تواند کیفیت جواب‌های به‌دست آمده را افزایش دهد. شمارنده‌ی کارایی سخت افزاری[۱] ابزاری است که توسط مهندسین نرم‌افزار مورد استفاده قرار می‌گیرد تا بتوانند با تغییر کد، کارایی نرم افزار را بهبود دهند.

شمارنده کارایی سخت‌افزاری روی قسمت die[۲] در پردازنده قرار دارد. تعداد شمارنده‌های سخت‌افزاری موجود در هر پردازنده به نوع و مدل پردازنده بستگی دارد چون ممکن است در هر پردازنده رویدادهای متفاوتی وجود داشته باشد که برنامه‌نویسان بخواهند آن‌ها را شمارش کنند. برای مثال در AMD Athlon چهار شمارنده 40 بیتی وجود دارد.

شمارنده کارایی سخت‌افزاری دارای دو نوع ثبات است: ثبات‌های شمارش و ثبات‌های کنترل. این شمارنده‌ها می‌توانند تعداد رویدادها یا تعداد دوره‌های زمانی که شرط خاصی در آن برقرار است را اندازه‌گیری کنند. در هنگام شمارش رویدادها، شمارنده در زمان وقوع رویداد یک واحد افزایش پیدا می‌کند. ثبات‌های کنترلی طوری برنامه‌ریزی می‌شوند که بتوانند در صورت سرریزی شمارنده، شمارنده‌ی مربوطه، وقفه‌ای را ایجاد کند. همچنین شمارنده می‌تواند تعداد سیکل‌هایی را که شرط خاصی در آن برقرار می‌شود را شمارش کند. در AMD Athlon بیشتر از ۵۰ رویداد قابل شمارش است.

۲- رویدادهای کارایی[۳]

هر رویداد به معنای وقوع سیگنال خاص مرتبط با کارایی پردازنده است. شمارنده‌های کارایی سخت‌افزاری به عنوان مجموعه‌ی کوچکی از ثبات‌ها، این رویدادها را شمارش می‌کنند. مانند تعداد عدم دسترسی به حافظه‌ی نهان و

[1] Hardware Performance Counter (HPC)

[۲] یک لایه‌ی سیلیکنی که بر روی CPU قرار می‌گیرد.

[3] Performance event

عملگرهای ممیز شناور، زمانی که برنامه روی پردازنده اجرا می‌شود. هر پردازنده شامل تعدادی رویداد محلی[1] است که متناسب با معماری آن پردازنده می‌باشد. رویدادهای محلی، مجموعه‌ای از تمامی رویدادهایی هستند که توسط پردازنده قابل شمارش می‌باشند.

۲-۱- نظارت بر رویدادهای مربوط به کارایی

رویدادهای مربوط به کارایی را می‌توان به پنج دسته تقسیم کرد:

۱. ویژگی‌های برنامه: این رویدادها کمک می‌کنند تا بتوانیم ویژگی‌های برنامه یا سیستم‌عامل را تعریف کنیم که کاملا از پیاده‌سازی پردازنده مستقل است. برای مثال از مشهورترین این رویدادها می‌توان به تعداد و نوع دستورات اشاره کرد که می‌تواند شامل تعداد دستورات بارگذاری، ذخیره‌سازی، محاسبات ممیز اعشاری و مسیر و انشعاب‌ها می‌شود.

۲. دسترسی به حافظه: این رویدادها اغلب به عنوان بزرگترین دسته‌بندی رویدادها به شمار می‌روند و به تحلیل‌گران کارایی سلسله مراتب حافظه کمک بسیاری می‌کنند. برای مثال در رویدادهای حافظه می‌توان مراجعه و یا عدم مراجعه به انواع مختلف حافظه‌های نهان و تراکنش‌ها بر روی باس حافظه‌ی پردازنده را شمارش کرد.

۳. PiplineStall: دستورالعملpipeline تکنیکی برای طراحی کامپیوترها است تا بتوانند از این طریق بازدهی را افزایش دهند. در این روش چرخه‌ی انجام دستورالعمل به مراحل موازی شکسته می‌شود، در نتیجه، دستورالعمل‌هایی که در گام‌های متفاوتی قرار دارند، می‌توانند به صورت موازی اجرا شوند. پس از این تعریف کوتاه، در محاسبات کامپیوتری، Stall یا حباب تاخیری در اجرای یک دستورالعمل است تا یک خطا[2] برطرف شود. خطا زمانی اتفاق می‌افتد که در pipeline، در چرخه‌ی زمانی جاری، دستورالعملی قابل اجرا نباشد و به صورت بالقوه جواب های اشتباه تولید کند. شمارش رویدادهای مربوط به pipeline stallها به کاربر کمک می کند تا بتواند جریان برنامه در pipeline را تحلیل کند.

[1] Native event
[2] Hazard

۴. پیش‌بینی انشعاب: در معماری کامپیوتر پیش‌بینی‌کننده‌ی انشعاب[1] یک مدار دیجیتالی است که سعی می‌کند یک انشعاب مانند دستور if–then-else را قبل از اینکه از وقوع آن مطمئن شود، پیش‌بینی کند. هدف پیش‌بینی‌کننده‌های انشعاب بهبود جریان در pipeline است. این مدارهای دیجیتالی نقش بسیار مهمی در بهبود کارایی پردازشگرها به خصوص پردازشگرهای x86 دارد. پردازشگرهایی که دارای pipeline عمیق هستند، برای اینکه بتوانند این pipelineها را با دستورات مناسب پر کنند، به شدت به این پیش‌بینی‌کننده‌ی انشعاب‌ها وابسته هستند. رویدادهای پیش‌بینی انشعاب، به کاربران امکان می‌دهند تا کارایی سخت‌افزار پیش‌بینی کننده‌ی انشعاب را اندازه بگیرند. برای مثال می‌توانند این کار را با شمارش انشعاب‌هایی که به اشتباه پیش‌بینی شده‌اند انجام دهد.

۵. مصرف منابع: رویدادهای استفاده از منابع به ما کمک می‌کنند تا بتوانیم مشخص کنیم که پردازنده چه مواقعی از یک منبع استفاده می‌کند. با این کار می‌توان از منابع به صورت بهینه استفاده کرد.

۳- سخت‌افزار ناظر بر کارایی

سخت‌افزار ناظر بر کارایی به دو بخش آشکارساز رویداد کارایی[2] و شمارنده‌های رویداد[3] تقسیم می‌شود. با تنظیم و پیکربندی این دو بخش کاربران قادر خواهند بود تا انواع مختلفی از رویدادهای کارایی را تحت شرایط مختلف به‌دست آورند.

کاربران می‌توانند با تنظیم آشکارساز، هر یک از رویدادهای گفته شده در فوق را شناسایی کنند. اغلب این آشکارسازها فیلدی با نام فیلد پوشش رویداد[4] دارند که اجازه می‌دهد تا شرایط و وضعیت‌های مختلف رویداد را بررسی کنیم. برای مثال در پردازنده پنتیوم III اینتل، برای شمارش تعداد دسترسی‌های بارگذاری به L2 cache، فیلد پوششی وجود دارد که به آشکارساز این اجازه را می‌دهد تا این دسترسی‌ها را در وضعیت‌های اصلاح شده[5]، انحصاری[6] یا نامعتبر[7]

[1] Branch Predictor
[2] Performance event detector
[3] Event counters
[4] Event mask field
[5] Modified
[6] Exclusive
[7] Invalid

نظارت کند.

همچنین آشکارساز رویداد به ما این امکان را می‌دهد تا نظارت‌های خود را بر روی سطح دسترسی[1] جاری پردازنده انجام دهیم. سیستم‌عامل از سطح دسترسی‌های کاربر[2] و ناظر[3] استفاده می‌کند تا به برنامه‌های کاربردی امکان دسترسی و دستکاری سخت‌افزار و داده‌های حیاتی تحت نظارت مستقیم سیستم‌عامل، داده نشود. زمانی‌که سیستم‌عامل در حال اجرا بر روی پردازنده است، سطح دسترسی در حالت ناظر قرار می‌گیرد و زمانی که برنامه کاربردی روی پردازنده اجرا می‌شود، سطح دسترسی به کاربر تغییر پیدا می‌کند. بنابراین هم این امکان وجود دارد که تنها رویدادهای ایجاد شده توسط سیستم‌عامل یا برنامه کاربردی شمارش شود و هم می‌توانیم آشکارساز رویداد را طوری تنظیم کنیم که تمام رویدادها در دو سطح کاربر و ناظر را شناسایی کند. البته بر اساس سطح دسترسی‌های تعریف شده در سیستم‌عامل افراد عادی ممکن است اجازه‌ی استفاده از شمارنده کارایی را نداشته باشند.

علاوه بر شمارش رویدادهایی که توسط آشکارساز رویداد کارایی شناسایی می‌شود، کاربر می‌تواند شمارنده رویداد کارایی[4] را طوری تنظیم کند تا تنها رویدادها را تحت شرایط معین و لبه‌ای شمارش کند که برای این کار از ویژگی تشخیص لبه[5] استفاده می‌شود تا حضور و عدم حضور برخی شرایط را در هر سیکل تشخیص دهد. برای مثال، می‌تواند حضور یا عدم حضور pipe stall را در هر سیکل نشان دهد در نتیجه شمارش این رویدادها، تعداد سیکل‌هایی را نشان می‌دهد که در آن pipe stall رخ داده است. با این حال، ویژگی تشخیص لبه می‌تواند به جای این‌که تعداد سیکل‌هایی که stall در آن روی داده شمارش کند، تعداد stall ها را هم محاسبه کند (به خصوص زمان‌هایی را که یک stall آغاز شده است).

دومین ویژگی اصلی شمارنده، پشتیبانی از آستانه است. این قابلیت به آشکارساز رویداد این امکان را می‌دهد که مقداری را که در هر سیکل گزارش می‌کند با مقدار آستانه مقایسه کند در صورتی که مقدار گزارش‌شده به آستانه رسیده باشد، شمارنده یک واحد زیاد می‌شود. این ویژگی برای آن دسته از رویدادهای کارایی مفید است که مقدار گزارش‌شده آن‌ها در هر سیکل از یک بزرگتر است. برای مثال، در پردازنده‌های Superscalar می‌توان بیش از یک دستور را در هر

[1] Privilege mode
[2] User
[3] Supervisor
[4] Performance event counter
[5] Edge detection feature

سیکل تکمیل کرد. انتخاب دستورات کامل‌شده به عنوان رویداد کارایی و تنظیم آستانه برابر دو، منجر می‌شود تا زمانی که سه دستور یا بیشتر در هر سیکل کامل شود، مقدار شمارنده یک واحد افزایش یابد و به ما این امکان را می‌دهد تا تعداد دفعاتی را که سه دستور یا بیشتر در هر سیکل کامل شده‌اند، محاسبه کنیم.

٤- مشخصه کارایی[1]

آشکارساز و شمارنده رویداد کارایی، به آسانی می‌توانند وجود مشکلات کارایی را تشخیص دهند و همچنین به کاربران این امکان را می‌دهند تا شدت مشکل را تخمین بزنند اما در اکثر مواقع لازم است تا مکان‌هایی از کد را مشخص کنیم (هم در برنامه کاربردی و هم در سیستم‌عامل) که این مشکلات را ایجاد کرده‌اند. دانستن منشا مشکل این امکان را به برنامه‌نویس می‌دهد تا الگوریتم‌های سطح بالا[2] و یا سطح پایینی[3] را که توسط برنامه استفاده می‌شود، اصلاح کرده تا از تاثیرات جانبی آن مشکل جلوگیری کند.

٤-١- مشخصه مبتنی بر زمان[4]

این روش تکنیک رایجی است که در آن درصد زمانی را که یک برنامه در بخش‌های اصلی خود می‌گذراند، تخمین می‌زند. تمرکز بر روی قسمت‌هایی از برنامه که بیشتر اجرا می‌شوند، بهبود کارایی را در انجام تغییرات این کدها افزایش می‌دهد. مشخصه مبتنی بر زمان، در بازه‌های زمانی مشخص وقفه‌هایی را در اجرای برنامه ایجاد می‌کند. در طول هر وقفه، روتین سرویس وقفه[5] مقدار شمارنده برنامه را ذخیره می‌کند. زمانی که برنامه کامل شد، کاربر می‌تواند هیستوگرامی را ایجاد کند که تعداد نمونه‌های جمع‌آوری شده برای هر شمارنده برنامه را نشان می‌دهد که نهایتا نواحی از برنامه را که بیشتر اجرا شده به کاربر ارائه خواهد داد.

[1] Performance profile
[2] high level
[3] Low level
[4] Time-based profile
[5] Interrupt Service Routine

٤-٢- مشخصه مبتنی بر رویداد[1]

روشی مشابه با مشخصه مبتنی بر زمان است که در آن هیستوگرام تعداد رویدادهای کارایی به عنوان تابعی از مکان کد رسم می‌شود و در آن به جای ایجاد وقفه در بازه‌های زمانی مشخص که در پروفایل مبتنی بر زمان انجام می‌شد، سخت‌افزار ناظر بر کارایی، بعد از رخداد تعداد معینی از رویدادها، در برنامه ایجاد وقفه می‌کند. بنابراین همان‌طور که قبلا گفته شد در پروفایل مبتنی بر زمان، دستوراتی را که غالبا اجرا می‌شوند نشان داده ولی در پروفایل مبتنی بر رویداد، دستوراتی را نشان می‌دهند که غالبا باعث ایجاد رویداد کارایی می‌شوند.

٥- بررسی شمارنده کارایی سخت‌افزاری

روش‌های دیگری نیز برای جمع‌آوری داده‌های مربوط به کارایی پردازنده‌ها وجود دارد که بر پایه سخت‌افزار ناظر بر کارایی نیستند. برای مثال در یکی از این روش‌ها، نیاز به اصلاح برنامه وجود دارد تا اطلاعات مربوط به اجرای خودش را جمع‌آوری کند. در این حالت، این قسمت از برنامه نیاز دارد یا خودش را از روی کد منبع برنامه مجدد بسازد[2] یا نسخه اجرایی خود را اصلاح کند. به این نسخه اصلاح شده کدهای دیگری نیز افزوده می‌شود تا اطاعات مختلف را جمع‌آوری کند که این اطلاعات معمولا شامل ردیابی دستورات[3] و مراجعات به حافظه می‌باشد. سپس ابزارهای دیگری این داده‌های تولید شده را پردازش کرده تا داده‌های مربوط به کارایی را تولید کنند. با این حال گاهی اوقات استفاده از این دیدگاه مشکل است چون همیشه امکان ساختن مجدد برنامه وجود ندارد. برای مثال امکان دارد که کد منبع در دسترس نباشد. همچنین این دیدگاه‌ها می‌توانند رفتار برنامه را مختل کنند که در نتیجه اعتبار داده‌های جمع‌آوری شده را زیر سوال می‌برد.

استفاده از شبیه‌سازی که پردازنده را مدل کرده و برنامه را اجرا کند، راهکار دیگری است که از طریق آن می‌توان داده‌های مربوط به کارایی (از قبیل داده‌های مربوط به pipeline stall، پیش‌بینی انشعاب، کارایی حافظه‌ی نهان و غیره) را به‌دست آورد. با این حال سازندگان پردازنده‌ها، جزئیات اطلاعات ساخت و پیاده‌سازی پردازنده‌های خود را فاش نمی‌کنند، در نتیجه توسعه این شبیه‌سازها بدون این اطلاعات مشکل خواهد بود. با این حال اگر این شبیه‌ساز در اختیار

[1] Event-based profile
[2] Re-build
[3] Instruction trace

باشد، دقت کارایی آن همواره زیر سوال خواهد بود و سرعت این شبیه‌ساز به میزان قابل توجهی از اجرای برنامه و جمع‌آوری داده بر روی پردازنده واقعی کمتر خواهد بود.

استفاده از سخت‌افزار ناظر بر کارایی مزایای متعددی نسبت به روش‌های دیگر دارد. در این روش به دلیل این‌که خود پردازنده داده‌های مربوط به کارایی برنامه را هنگام اجرا جمع‌آوری می‌کند، از مشکلات متعددی جلوگیری می‌شود:

- عدم نیاز به شبیه‌سازی برنامه، سیستم‌عامل و یا پردازنده اعتبار و دقت شمارنده‌های رویداد جمع‌آوری شده را تضمین می‌کند.
- در دیدگاه سخت‌افزار ناظر بر کارایی داده‌ها در هنگام اجرا برنامه جمع‌آوری می‌شوند که باعث می‌شود جمع‌آوری رویدادهای کارایی با سرعت بالایی انجام شود که در مقابل دیدگاه مبتنی بر شبیه‌سازی قرار می‌گیرد.
- در این دیدگاه امکان جمع‌آوری داده‌های مربوط به برنامه و سیستم‌عامل وجود دارد.

از طرفی نظارت بر روی شمارنده کارایی سخت‌افرای با محدودیت‌هایی روبرو است که می‌توان از آن جمله به موارد زیر اشاره کرد:

- کم بودن تعداد شمارنده‌ها : پردازشگرها معمولا تعداد کمی شمارنده کارایی دارند که می‌توانند به صورت موازی اجرا شوند. بنابراین جمع‌آوری اطلاعات شمارنده‌ها برای رویدادهای مختلف مستلزم این است که برنامه بارها و بارها اجرا شود.
- شمارش speculative: برای پردازشگرهایی که رویدادهای کارایی را قبل از تمام شدن دستورالعمل کشف می‌کنند، مشکل پیش‌بینی speculative به وجود می‌آید. تعداد زیادی از پردازشگرهایی که کارایی بالایی دارند، برای این کار باید pipeline های عمیقی داشته باشند. درحالی‌که شاخه‌ها در برنامه‌ها مسیر اجرای برنامه را از حالت عادی منحرف می‌کنند. در این حالت، زمانی‌که پردازشگر pipeline را flush[1] می‌کند و دستور درست را واکشی می‌کند، کاهش کارایی قابل ملاحظه‌ای مشاهده می‌شود. برای کاهش دادن فرکانس وقوع pipeline flush، اکثر پردازنده‌ها از پیش‌بینی‌کننده‌های انشعاب استفاده می‌کنند که خروجی انشعاب را پیش‌بینی می‌کند. درنتیجه این پردازنده تعداد زیادی از دستورالعمل‌ها را به صورت speculative پیش‌بینی

[1] Pipeline دارای پنج مرحله است که مرحله‌ی آخر flush کردن و شروع دستورالعمل جدید است.

می‌کنند. برخی از این پیش‌بینی‌های speculative به مسیر اجرای نادرستی اشاره می‌کنند و هرگز اجرای آن‌ها کامل نمی‌شود. دستورالعمل‌هایی که در مسیر درست قرار دارند کامل اجرا می‌شوند. به دلیل این که دستورالعمل‌هایی که اجرای آنها فقط پیش‌بینی شده است هم باعث تغییر در مقادیر شمارنده‌های کارایی می‌شود لازم است که این دستورالعمل‌ها از دستورالعمل‌هایی که واقعا اجرا می‌شوند جدا شود.

- تاخیر نمونه‌برداری: مشکل بزرگی که در سخت‌افزارهای ناظر بر کارایی وجود دارد حمایت نادرست EBS ها است. EBS ها این اجازه را می‌دهند تا بتوان مجموعه‌ای از دستورالعمل را پیدا کرد که بیشتر از دستورالعمل های دیگر باعث وقوع رویداد خاصی شده است. وقتی این دستورالعمل شناخته شد، برنامه‌نویس می‌تواند کد مربوط به آن دستورالعمل را بازنویسی کند تا به این ترتیب کم شدن کارایی را جبران کند. اکثر سخت‌افزارهای EBS بر یک شمارنده‌ی کارایی خاصی نظارت می‌کنند و در صورت سرریز شدن شمارنده به یک PMI[1] سیگنالی را ارسال می‌کنند. سپس روتین سرویس وقفه‌ی مربوط به PMI مقدار شمارنده را برای دستوری که وقفه را ایجاد کرده است ثبت می‌کند. به هر حال این دستورالعمل معمولا همان دستورالعملی نیست که وقفه را ایجاد کرده است. pipeline پردازنده و تاخیر بین سرریز شمارنده و سیگنالینگ یک وقفه با هم ترکیب شده و بین دستورالعملی که رویداد را ایجاد کرده و دستورالعملی که مقدار روتین سرویس وقفه‌ی PMI را ثبت کرده است فاصله‌ی تصادفی ایجاد می‌کند. برای مثال فرض کنید یک pipeline با ۱۵ گام داریم و دستورالعملی که در گام ۵ قرار دارد باعث سرریز شدن بافر شده است. اگر رویداد، بلافاصله یک وقفه را ایجاد کند، حداقل ۱۰ دستور قبلی در pipeline باقی می‌ماند. حال، روتین سرویس وقفه‌ی PMI ممکن است به اشتباه تشخیص دهد که دستورالعمل قبلی دستورالعملی است که این وقفه را ایجاد کرده است. این در حالی است که اگر این وقفه در گام آخر ایجاد می‌شد روتین سرویس وقفه‌ی PMI این دستورالعمل را می‌توانست به درستی تشخیص دهد. این مورد باعث می‌شود که EBS ها نتوانند دستورالعملی که در کارایی مشکل ایجاد کرده است را تشخیص دهند.
- کمبود Data-addressed profiling: اکثر سخت‌افزارهای ناظر بر کارایی موجود، مشکل عدم جمع‌آوری پروفایل داده های مورد استفاده توسط برنامه‌ای که رویداد مربوط به کارایی را ایجاد کرده است را دارند.

[1] Performance Monitor Interrupt

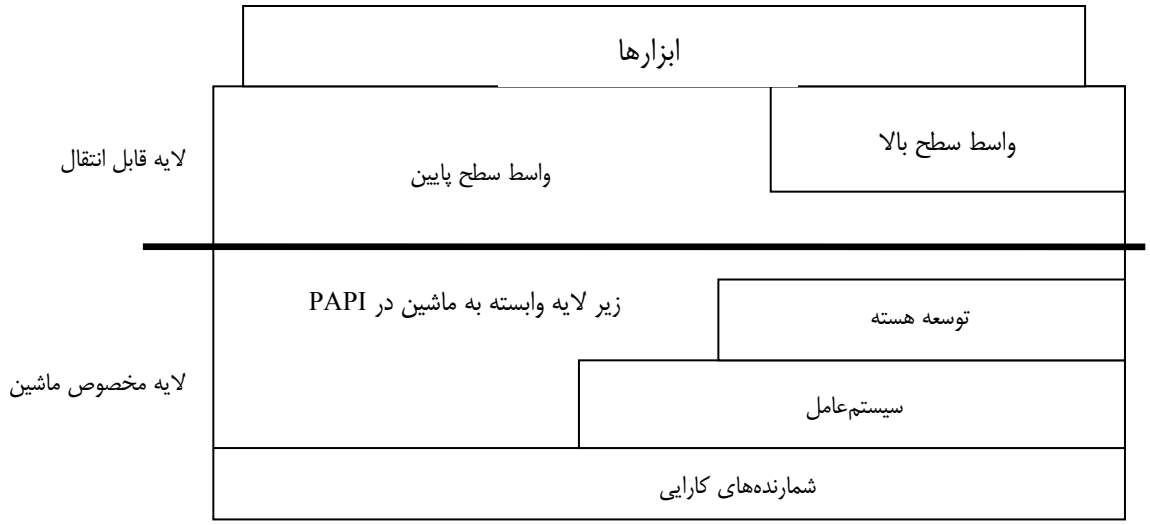
٦- معرفی PAPI

پروژه‌ی PAPI واسط برنامه‌نویسی کاربردی[1] استانداردی را برای دسترسی به شمارنده‌های کارایی سخت‌افزاری موجود در اکثر پردازنده‌های امروزی فراهم می‌کند. این شمارنده‌ها به عنوان مجموعه‌ی کوچکی از ثبات‌ها هستند که رویدادهایی را شمارش می‌کنند که در سیگنال‌های خاصی از کارایی برنامه اتفاق می‌افتد. نظارت بر روی این رویدادها، ارتباط بین ساختار کد و کارایی سخت‌افزاری که کد بر روی آن اجرا می‌شود را آسان می‌کند. این ارتباط کاربردهای متنوعی در تحلیل کارایی دارد که می‌توان به تنظیم دستی، بهینه‌سازی کامپایلر، اشکال‌زدایی برنامه، نظارت و مدل کردن کارایی، بررسی سیستم یا برنامه اشاره کرد. PAPI برای سخت‌افزار شمارنده‌ی کارایی نیاز به دو واسط دارد. یک واسط سطح بالا و ساده تا بتوان از آن برای به دست آوردن مقادیر حاصل از اندازه‌گیری‌های روتین و قابل برنامه‌ریزی استفاده کرد. این واسط سطح بالا به سادگی توانایی شروع، پایان و خواندن رویدادهای خاصی را در یک زمان دارد. درحالی‌که واسط دوم که واسط سطح پایین است، برای کاربرانی کاربرد دارد که نیازمندی‌های پیچیده‌تری دارند. این واسط سطح پایین به رویدادهایی مربوط می‌شود که در گروهی به نام EventSetها قرار دارند. EventSetها مشخص می‌کنند که شمارنده‌ها اغلب به چه صورتی مورد استفاده قرار می‌گیرند. به عنوان مثال، اندازه‌گیری‌های همزمان رویدادهای سخت‌افزاری مختلف و مربوط کردن آنها به یکدیگر. به عنوان مثال، مربوط کردن سیکل‌های مراجعات حافظه می‌تواند مدیریت و مکان‌یابی ضعیف حافظه را نشان دهد. EventSetها کاملا قابل برنامه‌ریزی هستند و ویژگی‌هایی از قبیل تضمین صحت نخ‌ها[2]، نوشتن مقادیر شمارنده، اطلاع‌رسانی به‌هنگام عبور از حد آستانه را دارند.

شکل زیر طراحی داخلی معماری PAPI را نشان می‌دهد، این معماری شامل دو لایه می‌شود.

- لایه‌ی قابل انتقال که شامل دو لایه‌ی سطح پایین و سطح بالا و توابع مستقل از ماشین می‌باشد.
- لایه‌ی مخصوص ماشین که واسطی را برای تبدیل توابع مستقل از ماشین به توابع و ساختمان‌های داده‌ی وابسته به ماشین تعریف می‌کند. این توابع در لایه‌ی زیرین تعریف می‌شوند، که از فراخوانی‌های سیستم‌عامل، توسعه[3] هسته و زبان اسمبلی برای دسترسی به شمارنده‌های کارایی سخت‌افزاری استفاده می‌کنند. PAPI از بین موارد گفته شده، موثرترین و انعطاف‌پذیرترین آنها را انتخاب می‌کند.

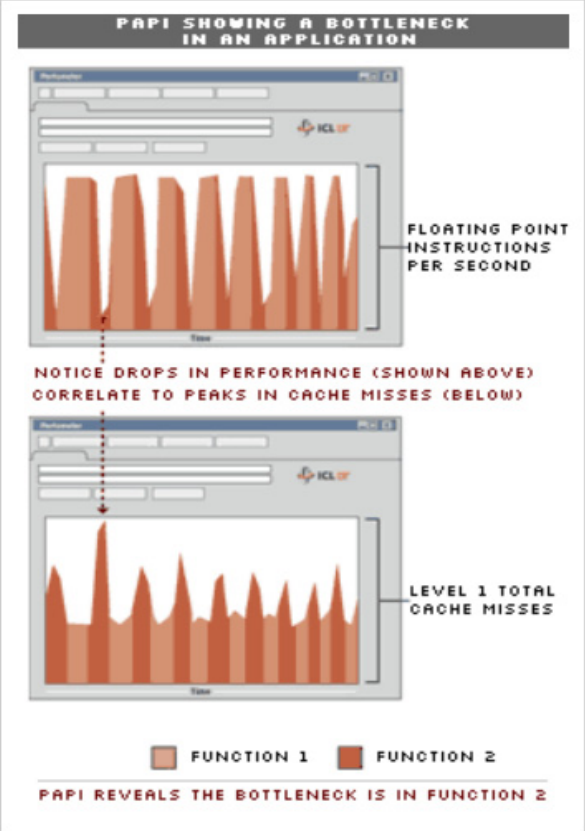
1 Application Programming Interface
2 Thread
3 Extension



شکل ۱: طراحی معماری داخلی PAPI

PAPI تلاش می‌کند تا یک محیط واحدی را برای تمامی پلتفرم‌های مختلف ارائه دهد. با این حال، همواره این ویژگی، امکان‌پذیر نیست. جایی که پردازنده‌ها برخی پارامترها را پشتیبانی نمی‌کنند، امکان نظارت بر روی رویدادهای مختلف وابسته به آن پارامترها امکان‌پذیر نیست. بنابراین واسط ثابت می‌ماند، اما نوع پیاده‌سازی آن در هر پردازنده می‌تواند متفاوت باشد.

PAPI به مهندسان نرم‌افزار این اجازه را می‌دهد تا تقریبا به صورت بی‌درنگ ارتباط بین کارایی نرم‌افزار و رویدادهای پردازنده را مشاهده کنند. ارتباط بین دو عمل در شکل ۲ نشان داده شده است. یکی از این رویدادها مربوط به فراخوانی خط مشخصی از کد منبع برنامه و دیگری عدم موفقیت در مراجعه به حافظه‌ی‌نهان مربوط به پردازنده است. به همین ترتیب PAPI می‌تواند کارایی و رابطه‌ی بین رویدادهای مختلف دیگر را هم با استفاده از شمارنده‌ی سخت‌افزار پردازنده به دست آورد.



شکل۲: ارتباط بین دو رویداد مختلف

همان‌طور که در شکل ۲ نشان داده شده است، نمودار اول مربوط به دستورات ممیز شناوری[1] است که در واحد زمان توسط پردازنده انجام می‌شود و نمودار دوم مربوط به عدم موفقیت در دسترسی به حافظه نهان سطح یک است. به دلیل افزایش تعداد عدم دسترسی به حافظه نهان، تعداد دستورات قابل انجام کم شده است. این مورد در کل نمودار برقرار است، یعنی جایی که تعداد عدم دسترسی‌ها کم باشد تعداد دستورات انجام شده در واحد زمان بیشتر خواهد بود. PAPI بین پلتفرم‌های مختلف قابلیت انتقال دارد و برای معماری‌های مختلف از همان روتین‌ها با لیست آرگومان‌های مشابه برای کنترل و دستیابی به شمارنده‌ها استفاده می‌کند.

به عنوان بخشی از PAPI، مجموعه‌ای از رویدادهای از پیش تعریف شده‌ای وجود دارد که شامل رویدادهایی می‌شود که در همه‌ی پیاده‌سازی‌های معتبر مربوط به شمارنده‌ها وجود دارد. هدف این است که شمارنده‌ها برای کدهای منبع مشابه، رویدادهای مشابه و قابل مقایسه‌ای را روی پلتفرم‌های مختلف بشمارند. اگر برنامه‌نویس از همین

[1] Float Point

رویدادهای استاندارد استفاده کند، نیازی به تغییر کد منبع نخواهد داشت. در حالی که اگر بخواهد به رویدادهای خاصی از ماشین دسترسی پیدا کند، API سطح پایین به او این اجازه را می‌دهد که همه‌ی رویدادهای ممکن را شمارش کند. وقتی یک رویداد یا یک ویژگی خاص بر روی پلتفرم جاری وجود نداشته باشد، PAPI کد خطای مناسبی را برمی‌گرداند. این ویژگی به طور قابل توجهی تلاش فراخوانی کد توسط PAPI را کاهش می‌دهد و فقط لازم است لیست آرگومان‌های فراخوانی، به روزرسانی شود.

برخی پردازنده‌ها مانند پردازنده‌های سری POWER دارای گروه‌های شمارنده هستند. PAPI قادر است به جای دسترسی به یک رویداد به گروه‌های خاصی از شمارنده‌ها دسترسی پیدا کند. البته این مساله باعث مطرح شدن مشکل جدی در ویژگی قابلیت انتقال می‌شود که PAPI از راهکارهایی برای حل این مشکل استفاده می‌کند.

PAPI دارای دو لایه‌ی نرم‌افزاری است. لایه‌ی بالا، یک API دارای توابع مستقل از ماشین است. لایه‌ی پایین، واسطی را برای تبدیل توابع لایه‌ی بالا به توابع و ساختمان داده‌های وابسته به ماشین ارائه می‌کند. توابع موجود در لایه‌ی پایین به زیرلایه‌هایی مانند سیستم‌عامل یا توابع اسمبلی دسترسی پیدا می‌کند تا بتواند به طور مستقیم با ثبات‌های پردازنده ارتباط برقرار کند. در واقع، کارایی لایه‌ی بالاتر به شدت به لایه‌ی زیرین وابسته است. درصورتی‌که لایه‌ی زیرین نتواند ویژگی‌های مطلوب را ارائه دهد، PAPI سعی می‌کند به شکل زیر مشابه آنها را فراهم کند. فرض PAPI بر این است که سیستم‌عامل یا کتابخانه، از سرریز شدن مقادیر شمارنده محافظت می‌کند. هر شمارنده، این توانایی را دارد که چندین بار در هر سیکل افزایش پیدا کند.

یکی از ویژگی‌های پیشرفته‌ی PAPI، این است که در صورت رسیدن مقدار شمارنده به حد آستانه‌ای که کاربر آن را مشخص کرده است، اطلاع‌رسانی ناهمگامی را فراهم می‌کند. در نتیجه، این باعث می‌شود که یک هیستوگرام دقیقی از کارایی ایجاد شود که مبتنی بر زمان نیست.

۷- نتیجه‌گیری

به طور کلی، از شمارنده کارایی سخت‌افزاری می‌توان برای تحلیل برنامه، سیستم‌عامل و کارایی پردازنده استفاده کرد. همچنین تحلیل‌های به‌دست آمده از این شمارنده‌ها می‌تواند به بهبود طراحی کامپایلرها و پردازنده‌های نسل جدید کمک بسیاری کند.

از طرفی پژوهش‌های جدید نشان می‌دهد که با افزایش تعداد کامپیوترها، تعداد بدافزارها نیز افزایش می‌یابد. سیستم‌های امروزی تنوع مختلفی از انواع ویروس‌ها، روت‌کیت‌ها و جاسوس‌افزارها[1] و دسته‌های دیگری از بدافزارها را در خود دارد. برخلاف اینکه تعداد زیادی از نرم‌افزارهای ویروس‌یاب وجود دارد، بدافزارها در پیدا کردن راهی برای نفوذ به سیستم پافشاری بیشتری می‌کنند. در واقع می‌توان گفت مهاجمین با پیدا کردن خطا در نرم‌افزار ویروس‌یاب، راه‌های نفوذ به سیستم را پیدا می‌کنند. طی پژوهش‌های انجام شده، نشان داده شده است که می‌توان توسط شمارنده‌های کارایی سخت‌افزاری بدافزارها را تشخیص داد. در صورت استفاده از شمارنده‌های سخت‌افزاری، می‌توانیم به دقت بالاتری و خطای کمتری در تشخیص بدافزارها دست پیدا کنیم چون این شمارنده‌ها در لایه زیرین نرم‌افزار قرار دارند و به طور مستقیم با سخت‌افزار در ارتباط هستند و احتمال خطا و دور زدن آن توسط بدافزار بسیار کم‌تر است.

مراجع

[1] Brinkly Sprunt, *The basics of performance monitoring hardware*, Journal of IEEE Micro, Volume 22 Issue 4, Page 64-71, July 2002.
[2] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, Salvatore J. Stolfo, *On the feasibility of online malware detection with performance counters*, In proceeding of 40th Annual International Symposium on Computer Architecture, Pages 559-570, 2013.
[3] Leif Uhsadel, Andy Georges, Ingrid Verbauwhede, *Exploiting Hardware Performance Counters*, 2010.
[4] Jack Dongarra, Kevin London, Shirley Moore, Phil Mucci, Dan Terpstra, *Using PAPI for Hardware Performance Monitoring on Linux*, In Proceedings of the 13th international conference on Information hiding, 2011.
[5] Glen Ammons, *Exploiting hardware performance counters with flow and context sensitive profiling*, In proceeding of the ACM SIGPLAN 1997 Conference Programming language design and implementation, Pages 85-96, 1997.
[6] D. Terpstra, H. Jagode, H. You, J. Dongarra,.*Collecting Performance Data with PAPI-C*, Tools for High Performance Computing 2009, Springer Berlin / Heidelberg, 3rd Parallel Tools Workshop, Dresden, Germany, pp. 157-173, 2009.

[1] Spyware