مدیریت حافظه چرک‌نویس و نگاشت وظایف در سامانه‌های بحرانی-مخلوط چندهسته‌ای

سینا عرب‌زاده[۱]، مرتضی محجل کفشدوز[۲] و عبدالرضا رسولی کناری[۳]
[۱]دانشگاه صنعتی قم، sinaarab@live.com
[۲]دانشگاه صنعتی قم، mohajjel@ce.sharif.edu
[۳]دانشگاه صنعتی قم، rasouli@qut.ac.ir

چکیده – هزینه قابل‌توجه بخش‌های پردازشی در سامانه‌های ایمنی-بحرانی باعث شده است که امروزه در صنایع مختلف مانند هواپیما و اتومبیل تمایل به تجمیع کردن این بخش‌ها بر روی تعداد کمتر هسته‌های پردازشی بیشتر شود. چالشی که با تجمیع کردن وظایف مختلف پردازشی بر روی هسته‌های پردازشی کمتر ایجاد می‌شود، سطح مختلف بحرانیت و تضمین موردنیاز هر یک از این وظایف است که به همین خاطر از آن‌ها به‌عنوان سامانه‌های بحرانی-مخلوط یاد می‌شود. یکی از منابع مهم در این سامانه‌ها حافظه‌ی روی تراشه است که بنا به دلایل پیش‌بینی‌پذیری در بسیاری از پردازنده‌ها به‌صورت حافظه چرک‌نویس پیاده‌سازی می‌شود. در این مقاله دو روش ارائه خواهیم کرد که هدف آن‌ها مدیریت توأمان حافظه چرک‌نویس، نگاشت وظایف به هسته‌های پردازشی، کاهش تعداد هسته‌های پردازشی و درنتیجه کاهش هزینه سامانه است. آزمایش‌ها انجام‌شده بر روی مجموعه وسیعی از محک‌ها نشان‌دهنده کارایی روش‌های ارائه شده است.

کلید واژه- پردازنده‌های چندهسته‌ای، حافظه چرک‌نویس، زمان‌بندی و نگاشت وظایف، سامانه‌های بحرانی-مخلوط.

۱- مقدمه

بخش‌های الکترونیکی در بسیاری از سامانه‌های ایمنی- بحرانی مانند هواپیما و اتومبیل دارای هزینه قابل توجه‌ای می‌باشد. برای مثال در یک خودرو *BMW* سال ۲۰۰۶ حدود صد ریزپردازنده استفاده شده است [۱] که به وظایفی مانند تنظیم سوخت‌رسانی مناسب به موتور، تنظیم نور چراغ‌ها، کنترل سامانه تهویه خودرو و تشخیص وضعیت راننده می‌پردازند.

یکی از روش‌هایی که برای کاهش هزینه بخش‌های الکترونیکی استفاده می‌شود مجتمع کردن بخش‌های کنترلی و پردازشی در پردازنده‌های با تعداد کمتر و به اشتراک گذاشتن منابع سخت‌افزاری است؛ برای این منظور استانداردهایی همچون استاندارد *AUTOSAR* برای صنعت خودروسازی و استاندارد *CAST32A* برای صنعت هواپیما ارائه شده است.

در سامانه‌های ایمنی- بحرانی به‌طور معمول بخش‌های مختلف سامانه، بحرانیت مختلفی دارند. بحرانیت یک بخش از سامانه بر اساس سطح تضمین مورد نیاز آن در برابر خرابی تعریف می‌شود [۲]. به سامانه‌هایی که بخش‌های مختلف آن‌ها سطح بحرانیت مختلفی دارند و مسئولیت پردازش این بخش‌های مختلف بر عهده یک سخت‌افزار است سامانه‌های بحرانی-مخلوط گفته می‌شود [۳].

اگرچه با تجمیع کردن بخش‌های مختلف کنترلی و پردازشی بر روی سخت‌افزار مشترک هزینه‌های سخت‌افزاری کاهش قابل توجه‌ای می‌یابند ولی چالش‌های بزرگی به وجود خواهد آمد. چالش اصلی این است که وظایف مختلف سامانه که هرکدام بحرانیت خاص خود رادارند و قبلاً بر روی پردازنده‌ها و منابع سخت‌افزاری جدا اجرا می‌شدند، اکنون باید بر روی یک سخت‌افزار اجرا بشوند.

در این سامانه‌ها برای ایجاد تضمین‌های لازم برای هر یک از وظایف با سطح بحرانیت مختلف، باید به نوعی بخش‌بندی بین وظایف با سطح بحرانیت مختلف به‌وجود آید؛ ولی در مقابل برای استفاده بهینه از سخت‌افزار باید منابع (مانند پردازنده‌ها، حافظه‌ها و غیره) به‌صورت مشترک استفاده بشوند.

یکی از منابع مشترک که می‌تواند تأثیر بسزایی در کارایی سامانه داشته باشد، حافظه‌های سطح تراشه هستند. این حافظه‌ها به دلیل سرعت بالایی که دارند می‌توانند به مقدار قابل توجه‌ای زمان اجرای وظایف را کاهش دهند. در پردازنده‌های عام‌منظوره این حافظه‌ها به‌صورت حافظه نهان پیاده‌سازی و کنترل آن توسط سخت‌افزار انجام می‌شود. نرم‌افزار به‌طور معمول کنترل مستقیمی بر محتوای حافظه نهان ندارد از این‌رو این حافظه از پیش‌بینی‌پذیری کمی برخوردار است لذا نمی‌توان به‌راحتی از آن در سامانه‌های بی‌درنگ استفاده کرد. برای حل این مشکل از حافظه چرک‌نویس [۴] استفاده می‌شود که توسط نرم‌افزار قابل دسترس و مدیریت است. علاوه بر این، حافظه چرک‌نویس ساختار ساده‌تری نسبت به حافظه نهان دارد و انرژی مصرفی آن نیز کمتر است [۵]. ولی مسئله‌ای که باید حل شود مدیریت کارایی حافظه چرک‌نویس است.

با توجه به اهمیت سامانه‌های بحرانی-مخلوط پژوهش‌های فراوانی در این زمینه انجام‌شده است. مقاله [۶] مرور جامعی بر این پژوهش‌ها داشته است. بیشتر پژوهش‌هایی که در این زمینه انجام شده‌اند به مسئله مدیریت حافظه چرک‌نویس توجه‌ای نکرده‌اند.

این مقاله به ارائه روش‌هایی برای نگاشت توأمان وظایف و مدیریت حافظه چرک‌نویس بر روی پردازنده‌های چندهسته‌ای می‌پردازد. هدف از این روش‌ها کاهش تعداد هسته‌های پردازشی استفاده شده و درنتیجه کاهش هزینه ساخت سامانه است. نوآوری‌های این پژوهش عبارت‌اند از:

۱. به رابطه متقابل زمان‌بندی وظایف، تخصیص حافظه چرک‌نویس و رفتار زمانی وظایف در سامانه‌های بحرانی-مخلوط توجه شده است.
۲. جهت کاهش تعداد هسته‌های پردازشی، یک مسئله‌ی بهینه‌سازی برای نگاشت وظایف و تخصیص حافظه چرک‌نویس طراحی‌شده است.
۳. با توجه به NP-کامل بودن مسئله مذکور [۵] روش‌های مکاشفه‌ای کارایی برای حل این مسئله ارائه شده است.

۴. روش‌های ارائه شده با مجموعه وظایف با ویژگی‌های مختلف آزمایش شدند. نتایج آزمایش‌ها نشان می‌دهد حتی در تعداد وظایف بالاتر روش‌های ارائه شده کارایی بالایی دارند و به‌سرعت به جواب‌های امکان‌پذیر و تعداد کم هسته‌های پردازشی (هدف بهینه‌سازی) همگرا می‌شوند.

در ادامه این مقاله ابتدا فرضیات، مدل در نظر گرفته شده و بحث‌های پایه توضیح داده خواهد شد. سپس به بررسی کارهای پیشین خواهیم پرداخت. مثال انگیزشی و روش‌های ارائه شده بخش‌های بعدی مقاله را تشکیل می‌دهند. درنهایت روش‌های ارائه شده و نتایج آزمایش آنها توضیح داده شده است.

۲- کارهای پیشین

در این بخش به بررسی پژوهش‌های پیشین در زمینه حافظه‌های چرک‌نویس و سامانه‌های بحرانی-مخلوط خواهیم پرداخت.

تخصیص حافظه چرک‌نویس در بسیاری از کارهای پیشین بررسی‌شده است. در مقاله [۷] یک الگوریتم تقریبی برای نگاشت وظایف و تخصیص حافظه چرک‌نویس ارائه شده است. هدف این الگوریتم در مقاله [۸،۷] کاهش تعداد جزیره‌ها در یک پردازنده چندهسته‌ای جزیره‌ای بوده است و در مقاله [۷] کاهش بهره‌وری هسته‌ها در یک پردازنده چندهسته‌ای با حافظه چرک‌نویس مشترک است. در مقاله [۵] یک روش پویا برای تخصیص حافظه چرک‌نویس برای سامانه‌های چندهسته‌ای نهفته بی‌درنگ ارائه شده است که در آن حافظه چرک‌نویس به‌صورت ایستا بین هسته‌ها تقسیم نمی‌شود و هر موقع یک وظیفه تمام می‌شود، فضای حافظه چرک‌نویس تخصیص‌یافته آن توسط سایر وظایف که به حافظه چرک‌نویس نیاز دارند مجدداً استفاده می‌شود. بعلاوه، در طرح پیشنهادی، تخصیص حافظه چرک‌نویس با زمان‌بندی و نگاشت وظایف برای رسیدن به کارایی بالاتر سامانه ادغام‌شده است. برای این منظور در این مقاله دو روش مبتنی بر برنامه‌ریزی خطی و الگوریتم ژنتیک جهت کاهش بدترین حالت زمان اجرا پیشنهادشده است. در مقاله [۹] به دلیل اینکه در هر زمان همه وظایف می‌توانند درخواست دسترسی به حافظه داشته باشند، بدترین حالت برای هر دسترسی لحاظ شده است که *WCET(Worse-cast exection time)* محاسبه‌شده را به‌شدت زیاد می‌کند، برای بهبود تخمین *WCET* در این مقاله برای دسترسی به گذرگاه از سیاست دسترسی چندگانه بخش زمانی (Time-division multiple access) *TDMA* استفاده شده است و این سیاست را در نوشتن مدل برنامه‌ریزی خطی و الگوریتم ژنتیک لحاظ کرده است. بدین ترتیب توانسته است سربار دسترسی به گذرگاه را در محاسبه *WCET* به مقدار قابل‌توجهی کاهش دهد. البته این کار باعث افزایش پیچیدگی مدل برنامه‌ریزی خطی می‌شود. مقاله‌های ذکرشده و سایر مقالات مشابه در زمینه تخصیص حافظه چرک‌نویس به مسئله بحرانی-مخلوط بودن سامانه توجه نکرده‌اند.

زمان‌بندی وظایف در سامانه‌های بحرانی-مختلط نیز در بسیاری از کارهای پیشین بررسی‌شده است. ولی بااین‌وجود هیچ‌کدام به مدیریت توأمان حافظه چرک‌نویس و نگاشت وظایف به‌صورت توأمان نپرداخته‌اند. مقاله [۶] بررسی جامعی در زمینه این پژوهش‌ها انجام داده است.

۳- مدل سامانه، فرضیات و پیش‌زمینه

در این بخش به توضیح درباره مدل سامانه و همچنین بحث‌های مقدماتی در مورد سامانه‌های بحرانی-مخلوط خواهیم پرداخت. فرض می‌کنیم یک سامانه بحرانی-مخلوط باید یک مجموعه متناهی از وظایف را اجرا کند. وظایف را متناوب در نظر می‌گیریم. هر وظیفه i، بر اساس دوره تناوب (فاصله بین دو زمان رهاسازی متوالی وظیفه)، موعد، زمان لازم برای اجرای وظیفه و سطح بحرانیت تعریف می‌شود که به ترتیب (از سمت چپ) با نمادهای $(T_i, D_i, \vec{C}_i, L_i)$ نشان داده می‌شوند. فرض می‌کنیم که موعدها با دوره تناوب برابر هستند $D_i = T_i$. این فرض در بسیاری از مقاله‌های معتبر در نظر گرفته شده است (برای مثال[۳]). توجه کنید که در سامانه‌های بحرانی-مخلوط برای وظایف در سطح بحرانیت مختلف زمان اجرای مختلفی در نظر گرفته می‌شود؛ بنابراین برداری از زمان‌های اجرا برای هر یک از وظایف داریم. با توجه به معادله (۱)، هر چه سطح بحرانیت بالاتر می‌رود زمان اجرای وظایف نیز بیشتر یا مساوی سطح بحرانیت پایین در نظر گرفته می‌شود:

$$L_1 > L_2 \rightarrow C(L_1) \geq C(L_2) \qquad (۱)$$

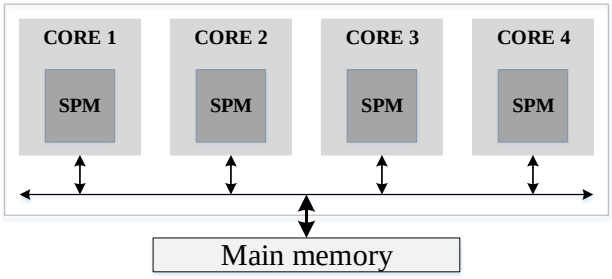
دلیل این امر این است که با بیشتر گرفتن بدترین زمان اجرای وظیفه می‌توان تضمین بیشتری برای اجرای صحیح آن داد.

در مدل در نظر گرفته شده از زمان‌بندی با اولویت ثابت استفاده می‌کنیم. از این نوع زمان‌بندی در سامانه‌های واقعی استفاده می‌شود. در این نوع زمان‌بندی در زمان طراحی به هر وظیفه یک اولویت بر اساس ویژگی‌های آن وظیفه داده می‌شود و این اولویت در زمان اجرا تغییر نمی‌کند. سامانه در هر زمان وظیفه آزادشده با بیشترین اولویت را به‌صورت قبضه‌ای (*Preemptive*) اجرا می‌کند. اگر در حین اجرای یک وظیفه یک وظیفه با اولویت بالاتر آزاد شود، اجرای وظیفه متوقف‌شده و پردازنده وظیفه با اولویت بالاتر را اجرا می‌کند.

مدل سخت‌افزار که ما در این مقاله در نظر گرفته‌ایم یک پردازنده چندهسته‌ای همگن با دو سطح حافظه می‌باشد:

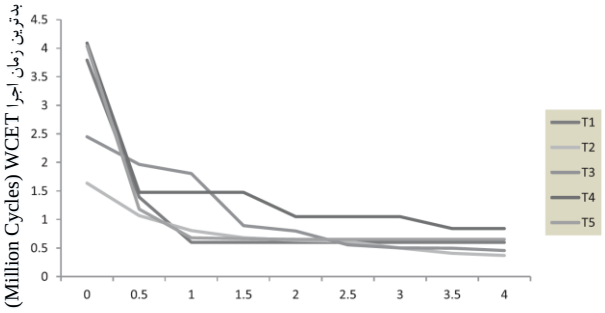
۱. حافظه‌های چرک‌نویس اختصاصی: هر هسته یک حافظه چرک‌نویس اختصاصی دارد که تنها توسط همان هسته قابل‌دسترس است.
۲. حافظه اصلی: که هسته‌ها به‌وسیله یک گذرگاه مشترک به آن دسترسی دارند.

قابل‌توجه است که در بسیاری از پردازنده‌های تجاری مانند *MPC5777M*، *TMS320C6472* و *ARM7TDMI* از حافظه چرک‌نویس استفاده شده است. شکل ۱ ساختار در نظر گرفته شده برای یک پردازنده چهار هسته‌ای را نشان می‌دهد. در ساختار در نظر گرفته شده، فرض شده است که هر هسته تأخیر بسیار کمی برای دسترسی به حافظه چرک‌نویس محلی خودش متحمل می‌شود [۱] و برای دسترسی به حافظه اصلی تأخیر بالاتری متحمل می‌شود. برای تضمین یک کران بالای ایمن برای زمان اجرا، ما فرض کرده‌ایم که هر دسترسی به حافظه خارج از تراشه موجب یک تأخیر ثابت (بدترین حالت) می‌شود.



شکل ۱: یک پردازنده چهار هسته‌ای با حافظه‌های چرک‌نویس SPM اختصاصی درون هر هسته

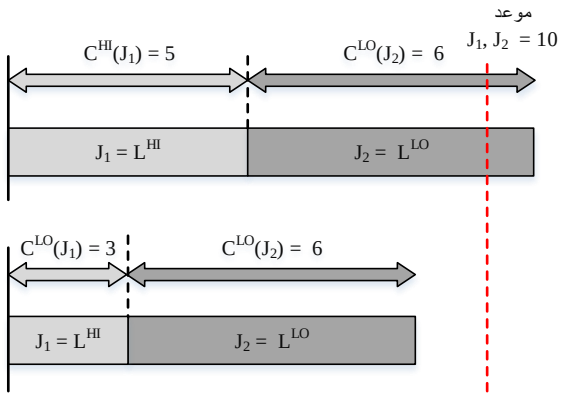
در مدل در نظر گرفته شده، می‌توان به هر وظیفه بخشی از فضای حافظه چرک‌نویس محلی را برای سرعت بخشیدن به اجرای آن تخصیص داد. افزایش سرعت به‌دست‌آمده برای هر وظیفه به‌منظور کاهش *WCET* آن، وابسته به مقدار فضای حافظه چرک‌نویس تخصیص‌یافته به آن می‌باشد. شکل ۲ نمودار بدترین زمان اجرای یکسری وظیفه بر اساس مقدار حافظه چرک‌نویس تخصیص داده‌شده را نشان می‌دهد.



شکل ۲: مقدار WCET یافته به ازای مقادیر متفاوت حافظه چرک‌نویس[۵].

برای اجتناب از مشکلات پایداری در حافظه‌های چرک‌نویس، فرض می‌کنیم که حداکثر یک کپی از هر شیء حافظه (منظور از شیء حافظه یک بخش از کد یا داده برنامه می‌باشد) می‌تواند در حافظه‌های چرک‌نویس وجود داشته باشد؛ بنابراین به علت اینکه ما چندین کپی از اشیای حافظه در حافظه‌های چرک‌نویس نداریم، مشکل هماهنگی به وجود نمی‌آید. همچنین برای مدیریت محتویات حافظه‌های چرک‌نویس به‌وسیله برنامه‌نویس یا کامپایلر، فرض می‌کنیم که هر حافظه چرک‌نویس دارای فضای آدرس غیرهمپوشان و مجزای خودش می‌باشد و این فضاهای آدرس در زمان طراحی معلوم هستند؛ بنابراین، کامپایلر یا برنامه‌نویس می‌تواند مستقیماً واحد حافظه‌ای (واحد حافظه چرک‌نویس یا حافظه اصلی) را که در آن هر شیء حافظه باید واقع شوند مشخص کنند.

در ادامه با یک مثال تئوری زمان‌بندی بحرانی-مخلوط را توضیح می‌دهیم. یک سامانه تک پردازنده قبضه‌ای را در نظر می‌گیریم که شامل دو کار J_1 و J_2 باشد که هر دو کار در زمان صفر در دسترس می‌باشند و باید در موعد زمانی ۱۰ خاتمه یابد. در شکل (۳) فرض کنید J_1 بحرانی حاد است و باید اتمام اجرای آن را قبل از موعد در سطح بالایی از اطمینان تضمین کرد و کار J_2 بحرانی سهل است و درنتیجه به سطح اطمینان بالایی نیاز ندارد. فرض می‌کنیم ابتدا J_1 سپس J_2 اجرا می‌شود. برای به دست آوردن $WCET$ کار J_1، با توجه به اینکه این وظیفه بحرانی حاد است، باید از ابزار تحلیل ایستا استفاده شود ولی برای وظیفه J_2 استفاده از تحلیل زمانی بر پایه اندازه‌گیری کافی است. فرض کنید $WCET$ ایستای کار J_1 برابر با ۵ و $WCET$ بر پایه اندازه‌گیری کار J_2 برابر با ۶ باشد. ازآنجاکه ۱۰ < ۶ + ۵، با استفاده از تکنیک‌های تجزیه‌وتحلیل زمان‌بندپذیری متعارف نتیجه‌گیری می‌شود که این سامانه قابل زمان‌بندی نیست و کار J_2 موعد را رعایت نمی‌کند؛ اما اگر به این نکته توجه کنیم که کار J_2 بحرانی سهل است و تضمین بالایی نیاز ندارد، می‌توان از تحلیل زمانی بر پایه اندازه‌گیری برای کار J_1 استفاده کرد و $WCET$ کمتری را برای J_1 به دست آوریم. فرض کنید نتیجه تحلیل زمانی بر پایه اندازه‌گیری مقدار ۳ را برای کار J_1 به دست آورد. در این صورت چون ۱۰ > ۶ + ۳ است، می‌توان (البته با سطح اطمینان پایین‌تری) تضمین کرد که کار J_2 نیز موعد را رعایت می‌کند [۱۰].



شکل ۳: تغییر مقدار WCET برای تضمین سطح بحرانیت یک کار

در مدل در نظر گرفته شده مانند بسیاری از کارهای پیشین (مانند[11], [3], [۱۲]) دو سطح بحرانیت را در نظر گرفته‌ایم: سطح بحرانیت بالا و پایین. سامانه در دو حالت کار می‌کند: در حالت عادی یا بحرانیت پایین و حالت بحرانیت بالا. در حالت بحرانیت پایین هیچ وظیفه‌ای از بدترین زمان اجرای بحرانیت پایین خود تجاوز نمی‌کند و همه وظایف بر اساس اولویت اجرا می‌شوند. سامانه بر مدت‌زمان اجرای وظایف نظارت می‌کند و درصورتی‌که وظیفه‌ای با سطح اولویت پایین از بدترین زمان اجرای بحرانیت پایین خود تجاوز کند اجرای آن وظیفه را متوقف می‌کند ولی اگر یک وظیفه با بحرانیت بالا از بدترین زمان اجرای بحرانیت پایین خود تجاوز کند سامانه اجازه این کار را می‌دهد و حالت سامانه از حالت عادی به بحرانیت بالا تبدیل می‌شود. در این حالت تنها وظایف با سطح بحرانیت بالا اجرا می‌شوند. البته در این حالت نیز بر زمان اجرای وظایف نظارت می‌شود و درصورتی‌که یک وظیفه با بحرانیت بالا از بدترین زمان اجرای بحرانیت بالای خود تجاوز کند اجرای آن متوقف می‌شود. برگشت به حالت عادی زمانی اتفاق می‌افتد که پردازنده بیکار باشد.

برای اینکه از اتمام اجرای وظایف قبل از موعدهایشان مطمئن شویم از تحلیل‌های زمان طراحی برای این منظور استفاده می‌شود. برای این منظور سامانه باید در سه حالت بررسی شود: در حالت نرمال، در حالت با بحرانیت بالا و حالت سویچ از حالت عادی به حالت با بحرانیت بالا. از سه رابطه بازگشتی زیر برای محاسبه زمان پاسخ هر یک از وظایف در حالت‌های مختلف استفاده خواهیم کرد:

$$R_i^{LO} = C_i(LO) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^{LO}}{T_j} \right\rceil C_j(LO) \qquad (۲)$$

$$R_i^{HI} = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^{HI}}{T_j} \right\rceil C_j(HI) \qquad (۳)$$

$$R_i^{*} = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^{*}}{T_j} \right\rceil C_j(HI) + \sum_{\tau_j \in hpL(i)} \left\lceil \frac{R_i^{LO}}{T_k} \right\rceil C_k(LO) \qquad (۴)$$

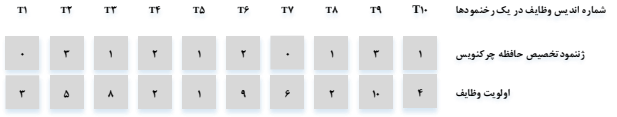
رابطه (۲) برای حالت نرمال سامانه، رابطه (۳) برای حالت بحرانیت بالا و رابطه (۴) برای حالت سویچ از حالت نرمال به حالت بحرانیت بالا استفاده می‌شود. قابل‌توجه است که برای تخصیص اولویت به وظایف از روش *AMC(Adoptive mixed-criticality)* استفاده خواهیم کرد. برای مطالعه بیشتر در زمینه سامانه‌های بحرانی، روابط تحلیل زمانی و روش *AMC* به مقاله [۳] مراجعه کنید.

۴- راه‌حل‌های پیشنهادی

این بخش به ارائه روش‌های پیشنهادی خواهیم پرداخت. به علت *NP*-کامل بودن مسئله، نمی‌توان آن را (حداقل با الگوریتم‌های موجود) در زمان چندجمله‌ای حل کرد. این موضوع زمانی اهمیت می‌یابد که پیچیدگی سامانه بیشتر شود که در این صورت زمان اجرای روش‌های بهینه بسیار زیاد خواهد شد. برای این منظور در این مقاله به ارائه روش‌هایی خواهیم پرداخت که اگرچه بهینه نیستند ولی می‌توانند جواب‌های مناسبی را در زمان‌های کم حتی برای سامانه‌های پیچیده به دست آورند. روش اول ارائه شده بر اساس الگوریتم فرا اکتشافی تبرید شبیه‌سازی شده [۱۳] و روش دوم بر اساس الگوریتم ژنتیک [۱۴] است. در ادامه به ترتیب به ارائه این دو روش می‌پردازیم.

۴-۱- الگوریتم ژنتیک

الگوریتم ژنتیک یک الگوریتم جستجوی فرا ابتکاری است که با موفقیت در چندین مسائله بهینه‌سازی، مانند برنامه‌ریزی و نگاشت وظایف [۱۵]–[۱۷] استفاده شده است. در این الگوریتم الهام گرفته شده از طبیعت، در شکل ۴ رخ‌نمودها (*Phenotypes*) به‌صورت رشته‌ها یا سایر ساختارهای داده‌ای ژن‌نمود (*Genotypes*) رمزگذاری شده‌اند. این الگوریتم با یک جمعیت اولیه از ژن‌نمودها آغاز می‌شود که یا به‌صورت تصادفی یا توسط یک روش ابتکاری ایجادشده است و این جمعیت به‌صورت تکراری تکامل می‌یابند تا رخ‌نمودهای بهینه یا نزدیک به بهینه پیدا شوند.



شکل ۴: ساختار یک رخ‌نمود که از دو ژن‌نمود تشکیل شده

برای این کار، سه عملیات انتخاب (*Selection*)، تقاطع (*Crossover*) و جهش (*Mutation*) که از طبیعت تقلیدشده است بر روی ژن‌نمودها انجام می‌شود. در عملیات انتخاب، برخی از ژن‌نمودها (معمولاً مناسب‌ترین) برای زنده ماندن در نسل بعدی و برای تولید ژن‌نمودهای جدید انتخاب می‌شوند. در عملیات تقاطع، دو یا چند ژن‌نمود از ژن‌نمودهای انتخاب‌شده برای تولید ژن‌نمودهای جدید ترکیب می‌شوند و درنهایت در عملیات جهش، یک یا چند داده از ژن‌نمودهای انتخاب‌شده برای حفظ تنوع ژن‌نمودها تغییر داده می‌شوند. این الگوریتم زمانی تمام می‌شود که یک شرط توقف مشخص، مانند حداکثر تعداد دورهای مجاز برآورده شود. برای اطلاعات بیشتر می‌توانید به کتاب‌هایی که در این زمینه نوشته‌شده است مانند [۱۴] مراجعه کنید.

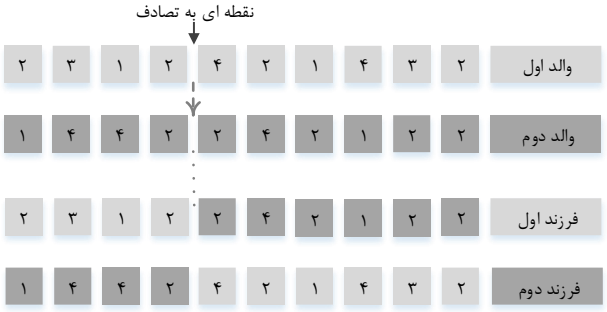
برای این‌که تخصیص حافظه چرک‌نویس و نگاشت وظایف را با استفاده از الگوریتم ژنتیک حل کنیم، باید روشی برای کد کردن هر رخ‌نمود به‌صورت یک ژن‌نمود مشخص کنیم. به دلیل اینکه یک رخ‌نمود برای این مسئله توسط دو نوع از اطلاعات (تخصیص حافظه چرک‌نویس/نگاشت وظایف) مشخص می‌شود، از دو رشته برای نشان دادن هر ژن‌نمود می‌توان استفاده کرد. ولی آزمایش‌هایی که انجام دادیم نشان داد این نحوه کد کردن مناسب نیست. به همین منظور به‌جای بخش نگاشت هسته از اولویت وظایف استفاده کردیم در اولین رشته، مانند [۵] تعداد بلوک‌های حافظه چرک‌نویس تخصیص داده‌شده به هر وظیفه مشخص می‌شود؛ در دومین رشته، اولویت وظایف در نگاشت به هسته‌های پردازشی مشخص می‌شود. برای تولید رخ‌نمود از یک ژن‌نمود وظایف بر اساس اولویت (رشته دوم) با استفاده از یک الگوریتم مانند (*first fit*) به هسته‌ها نگاشت می‌شوند.

برای مقایسه بین رخ‌نمودها باید برای هر رخ‌نمود یک مقدار برازندگی (*fitness*) تعریف شود. رابطه (۵) نحوه محاسبه برازندگی را برای رخ‌نمود p نشان می‌دهد.

$$fitness(p) = \begin{cases} \frac{1}{UsedCores(p)}, & p \in F \\ -UnScheduled(p), & p \notin F \end{cases} \quad (۵)$$

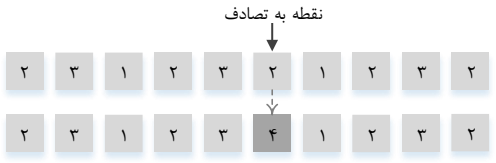
برای رخ‌نمودهایی که زمان‌بندپذیر (Schedulable) هستند ($p \in F$) مقدار برازندگی برابر با معکوس تعداد هسته‌های استفاده شده ($UsedCores(p)$) است. به این صورت که هرچقدر تعداد هسته‌های استفاده شده در یک رخ‌نمود کمتر باشد مقدار برازندگی آن بیشتر است. برای رخ‌نمودهایی که زمان‌بندپذیر نیستند ($p \notin F$) مقدار برازندگی کوچک‌تر از صفر و برابر با منفی تعداد وظایف زمان‌بندناپذیر ($-UnScheduled(p)$) خواهد شد.

شکل ۵ مثالی از عملیات تقاطع استفاده شده را نشان می‌دهد. ما از روش انجام ترکیب دو والدی با یک نقطه تقاطع استفاده کرده‌ایم. به این معنی که برای هر عملیات تقاطع یک نقطه در دو ژن‌نمود والد به‌صورت تصادفی انتخاب‌شده و تمامی اطلاعات در یک سمت از این نقطه بین دو والد تعویض شده و درنتیجه دو ژن‌نمود جدید (فرزندان) ساخته‌شده و به جمعیت دور بعد اضافه می‌شوند.



شکل ۵: عملیات تقاطع

شکل ۶ مثالی از عملیات جهش را نشان می‌دهد. در هر عملیات جهش یک داده از یک ژن‌نمود به‌صورت تصادفی انتخاب‌شده و مقدار آن به یک مقدار تصادفی معتبر جدید تغییر داده‌شده است.



شکل ۶: عملیات جهش

۴-۲- الگوریتم تبرید شبه‌سازی شده

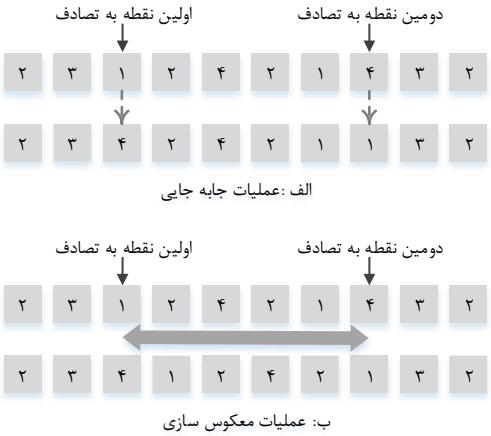
الگوریتم تبرید شبیه‌سازی شده که یک الگوریتم جست‌وجوی احتمالی برای حل مسائل بهینه‌سازی است که از فرآیند بازپخت در متالوژی الهام گرفته شده است. در این فرآیند ماده فیزیکی به‌آرامی سرد می‌شود تا درنهایت به یک پیکربندی با حداقل انرژی برسد. در الگوریتم تبرید شبیه‌سازی‌شده جواب مسئله را معادل باحالت ماده فیزیکی، هزینه یک جواب را معادل با انرژی ماده در یک حالت و پارامتر کنترلی را معادل با دما در نظر می‌گیرند. این الگوریتم از دو فرآیند تصادفی تشکیل می‌شود: یک فرآیند برای تولید جواب‌ها و یک فرآیند برای پذیرش جواب‌ها. در فرآیند تولید جواب از یک سری عملیات مانند جابه‌جایی (*Swap*) و معکوس (*Reversion*)، برای تولید جواب‌های همسایه به‌صورت تصادفی استفاده می‌شود. در فرآیند پذیرش جواب‌ها درصورتی‌که جواب همسایه تولیدشده هزینه کمتری داشته باشد حتماً به‌عنوان جواب بعدی انتخاب می‌شود و درصورتی‌که هزینه کمتری نداشته باشد احتمالاً به‌عنوان جواب بعدی انتخاب می‌شود. احتمال انتخاب جواب‌های با هزینه بیشتر در یک دور الگوریتم به دمای آن دور وابسته است و از تابع بولتزمن حساب می‌شود. با کاهش دما احتمال انتخاب جواب‌های بدتر نیز کاهش می‌یابد و درنتیجه احتمال انتخاب جواب‌های بدتر در دورهای نهایی نسبت به دورهای ابتدایی کمتر است. برای اطلاعات بیشتر می‌توانید به کتاب‌هایی که در این

زمینه نوشته‌شده است مانند [۱۳] مراجعه کنید.

برای حل مسئله تخصیص حافظه چرک‌نویس و نگاشت وظایف توسط این الگوریتم جواب‌های مسئله را با دو رشته مدل می‌کنیم. رشته اول مقدار حافظه چرک‌نویس تخصیص داده‌شده به وظایف و رشته دوم هسته‌ای را که هر یک از وظایف به آن نگاشت شده‌اند را مشخص می‌کند برای محاسبه هزینه جوابp از رابطه (۶) استفاده می‌کنیم که درواقع قرینه همان رابطه استفاده شده برای محاسبه برازندگی در بخش (۴-۱) است.

$$cost(p) = \begin{cases} -\frac{1}{UsedCores(p)}, p \in F \\ UnScheduled(p), p \notin F \end{cases} \quad (۶)$$

برای تولید جواب‌های همسایه یک جواب، از عملیات عوض کردن، معکوس کردن زیررشته و جهش استفاده می‌کنیم. در هر دور فقط یکی از این عملیات به‌صورت تصادفی انتخاب‌شده و انجام می‌شود در هریک از این عملیات ابتدا رشته مربوط به تخصیص حافظه چرک‌نویس و یا نگاشت هسته به‌تصادف انتخاب‌شده و سپس عملیات مربوطه انجام می‌شود. در عملیات عوض کردن دو اندیس به‌تصادف انتخاب‌شده و مقدار آن‌ها عوض می‌شود. در عملیات معکوس کردن زیررشته، دو اندیس در رشته جواب انتخاب‌شده و ترتیب داده‌ای بین آن معکوس می‌شود. شکل ۷ (الف و ب) هرکدام یک مثال از این عملیات را نشان می‌دهند. عملیات جهش مانند عملیات جهش (شکل ۶) در الگوریتم ژنتیک (بخش ۴-۲) یک اندیس به‌تصادف انتخاب و مقدار آن به‌تصادف (در محدوده مجاز) تغییر می‌کند.



الف :عملیات جابه جایی

ب: عملیات معکوس سازی

شکل ۷: دو روش به منظور تولید زیررشته همسایگی جدید

۵- آزمایشات

در این بخش به‌صورت تجربی با انجام یکسری آزمایشات به مقایسه روش‌های پیشنهادی خواهیم پرداخت و مزایا و معایب هر یک را بررسی خواهیم کرد.

۵-۱- راه‌اندازی آزمایشات

برای مقایسه روش‌های ارائه شده، هر دو روش ارائه شده در محیط *Visual Stduio* با زبان *C#* پیاده‌سازی شدند. مشابه باکارهای پیشین معتبر در زمینه سامانه‌های بحرانی-مخلوط (مانند[۱۸] [۳]، [۱۱]) یکسری وظیفه به‌صورت تصادفی با توزیع یکنواخت ایجاد شدند. پارامترهایی که به‌صورت تصادفی ایجاد می‌شوند عبارت‌اند از: بدترین زمان اجرای وظایف، دوره تناوب و سطح بحرانیت. برای در نظر گرفتن اثر تخصیص حافظه چرک‌نویس به وظایف اعداد تصادفی مشابه با مقاله [۵] ایجاد شدند. برای اینکه مقایسه بین دو روش جامع باشد، آزمایشات بر روی تعداد وظایف مختلف انجام شد (از ۱۰ الی ۲۰۰ وظیفه). همچنین برای مقایسه عادلانه بین دو روش همه آزمایشات بر روی یک سیستم با پردازنده *Corei7* و ۱۶ گیگابایت حافظه اجرا شدند.

پارامترهای در نظر گرفته شده برای هر یک از الگوریتم‌ها در جدول ۱ ذکرشده است.

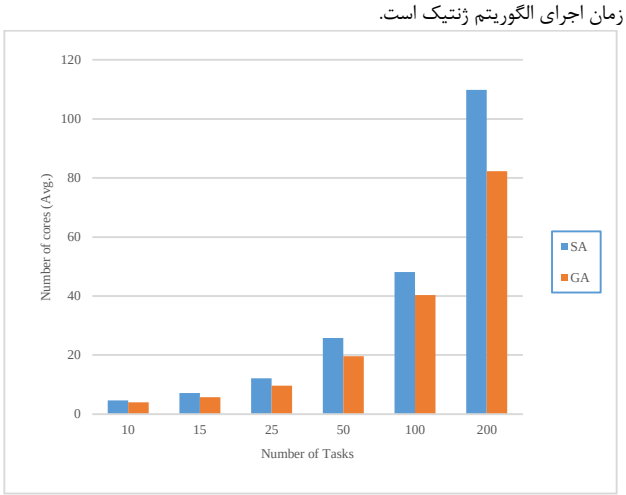
جدول ۱: پارامترهای تنظیم شده در اجرای روش‌های پیشنهادی

الگوریتم ژنتیک	احتمال عملیات جهش	۰.۵
	احتمال عملیات متقاطع	۰.۱
	تعداد جمعیت اولیه	۵۰
	حداکثر تعداد دور	۳۰۰
الگوریتم تبرید شبیه‌سازی	دمای اولیه	۲۰
	ضریب کاهش دمای تصاعد هندسی	۰.۹۹۸
	تعداد دور در یک دمای خاص	۵۰
	حداکثر تعداد دور	۲۰۰

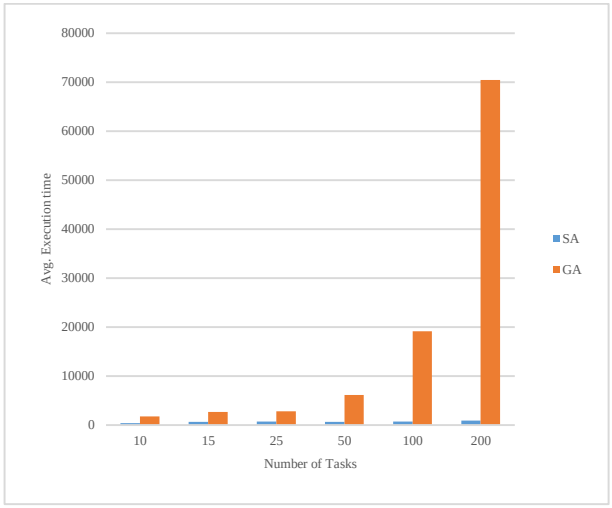
۵-۲- نتایج آزمایشات

نتایج به‌دست‌آمده از اجرای دو روش پیشنهادی برای مجموعه وظایف با تعداد مختلف را نشان می‌دهد. برای هر تعداد وظیفه آزمایش ۱۰ بار با مجموعه وظایف مختلف تکرار شده است و میانگین نتایج گزارش‌شده‌اند. همان‌طور که شکل ۸ نشان می‌دهد روش ژنتیک ارائه شده در حدود ۱۶ الی ۲۵ درصد بهتر از روش تبرید شبیه‌سازی‌شده عمل می‌کند.

شکل ۹ متوسط زمان اجرای هر یک از الگوریتم‌ها را برای تعداد وظایف مختلف نشان می‌دهد. همان‌طور که این شکل نشان می‌دهد، زمان اجرای هر یک از الگوریتم‌ها با افزایش تعداد وظایف افزایش می‌یابد. همچنین نتایج نشان می‌دهد که زمان الگوریتم تبرید شبیه‌سازی‌شده به‌طور خاص در تعداد وظایف بالا به‌طور قابل توجه‌ای کمتر از زمان اجرای الگوریتم ژنتیک است.



شکل ۸: تعداد هسته‌های پردازشی استفاده شده زمان‌بندپذیر به ازای هر تعداد وظایف



شکل ۹: متوسط زمان اجرای الگوریتم GA و SA به ازای هر تعداد وظایف

۶- نتیجه‌گیری

در این مقاله به مسئله مدیریت حافظه چرک‌نویس و همچنین نگاشت وظایف در سامانه‌های بحرانی-مخلوط چندهسته‌ای پرداختیم. برای این منظور دو روش بر اساس الگوریتم‌های فرا اکتشافی ژنتیک و تبرید شبیه‌سازی شده ارائه شد. برای ارزیابی این الگوریتم‌ها آزمایشات جامعی انجام شد. نتایج نشان داد که روش‌های ارائه شده می‌توانند ضمن حفظ تضمین‌های لازم برای وظایف با بحرانیت مختلف، تعداد هسته‌های استفاده شده و در نتیجه هزینه ساخت سامانه را تا حد خوبی کاهش دهند. برای کارهای آینده تعداد سطوح بحرانیت بیشتری را می‌توان در نظر گرفت. همچنین می‌توان روش‌های ارائه شده را به پردازنده‌های ناهمگون با مدل حافظه چند سطحی گسترش داد.

مراجع

[1] P. Marwedel, *Embedded system design : embedded systems, foundations of cyber-physical systems, and the internet of things*, 3rd ed. Cham: Springer International Publishing, 2018.

[2] S. Baruah, “Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations,” *IEEE Des. Test*, vol. 35, no. 2, pp. 31–37, 2018.

[3] S. K. Baruah, A. Burns, and R. I. Davis, “Response-time analysis for mixed criticality systems,” *Proc. - Real-Time Syst. Symp.*, pp. 34–43, 2011.

[4] E. Lee and S. Seshia, “Introduction to Embedded Systems: A Cyber-Physical Systems Approach,” 2016.

[5] M. M. Kafshdooz and A. Ejlali, “Dynamic shared SPM reuse for real-time multicore embedded systems,” *ACM Trans. Archit. Code Optim.*, vol. 12, no. 2, pp. 1–25, 2015.

[6] A. Burns and R. I. Davis, “Mixed Criticality Systems - A Review ∗ ,” no. March 2019, pp. 1–81.

[7] C.-W. Chang, J.-J. Chen, W. Munawar, T.-W. Kuo, and H. Falk, “Partitioned scheduling for real-time tasks on multiprocessor embedded systems with programmable shared srams,” in *Proceedings of the tenth ACM international conference on Embedded software - EMSOFT ’12*, 2012, p. 153.

[8] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, “Real-Time Task Scheduling on Island-Based Multi-Core Platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 538–550, Feb. 2015.

[9] D. Oehlert, A. Luppold, and H. Falk, “Bus-aware static instruction SPM allocation for multicore hard real-time systems,” *Leibniz Int. Proc. Informatics, LIPIcs*, vol. 76, no. 1, pp. 11–122, 2017.

[10] S. Vestal, “Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance,” in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.

[11] A. Burns, R. I. Davis, S. Baruah, and I. Bate, “Robust Mixed-Criticality Systems,” *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1478–1491, 2018.

[12] A. Burns and R. I. Davis, “Mixed Criticality Systems - A Review ∗ ,” pp. 1–52.

[13] K.-L. Du and M. N. S. Swamy, “Simulated Annealing,” in *Search and Optimization by Metaheuristics*, Cham: Springer International Publishing, 2016, pp. 29–36.

[14] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2003.

[15] F. A. Omara and M. M. Arafa, “Genetic algorithms for task scheduling problem,” *J. Parallel Distrib. Comput.*, vol. 70, no. 1, pp. 13–22, Jan. 2010.

[16] Y.-K. Kwok and I. Ahmad, “Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm,” *J. Parallel Distrib. Comput.*, vol. 47, no. 1, pp. 58–77, Nov. 1997.

[17] … M. G.-1999 D. A. C. (Cat. N. and undefined 1999, “Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system,” *ieeexplore.ieee.org*.

[18] D. Liu *et al.*, “Scheduling Analysis of Imprecise Mixed-Criticality Real-Time Tasks,” *IEEE Trans. Comput.*, vol. 67, no. 7, pp. 975–991, 2018.