علوم و مهندسی کامپیوتر
نشریه علمی پژوهشی انجمن کامپیوتر ایران
مجلد ۹، شماره ۱ و ۳ (الف)، ۱۳۹۰
صفحات ۳۹-۵۱
مقاله عادی

یک حل‌کننده عددی موازی برای زنجیره‌های مارکوف زمان‌پیوسته و پیاده‌سازی آن بر روی پردازنده‌های چندهسته‌ای

شهرزاد ترابی محمد عبداللهی ازگمی

دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران

چکیده

زنجیره‌های مارکوف زمان‌پیوسته (CTMCs)، برای تحلیل کارایی سیستم‌های کامپیوتری و ارتباطی مورد استفاده قرار می‌گیرند. با محاسبه احتمالات حالات پایدار CTMC، بسیاری از معیارهای مفید کارایی به دست می‌آیند. اما مدل‌های CTMC برای سیستم‌های واقعی بسیار بزرگ بوده، حل آنها زمان‌گیر است و از مشکل انفجار فضای حالت رنج می‌برند. در این مقاله یک راه‌حل موازی جدید برای حل حالت پایدار مدل‌های CTMC ارائه می‌کنیم. در این راه‌حل مشکل انفجار فضای حالت را با استفاده از موازی‌سازی روش‌های ضمنی تخفیف داده‌ایم. برای حل دستگاه معادلات از روش حل عددی ترکیبی جدیدی استفاده شده است که سرعت و دقت حل مدل را افزایش می‌دهد. ایده موازی‌سازی در سطح نرم‌افزار، با استفاده از الگوریتم پیشنهادی برای رنگ‌آمیزی گراف وابستگی وظایف و در سطح سخت‌افزار، با بهره‌گیری مؤثر از توان پردازنده‌های چندهسته‌ای پیاده‌سازی شده است. ابزار ساخته شده توسط محک‌های معروف در زمینه ابزارها و فنون حل مدل‌های مارکوفی، آزمایش و ارزیابی گردیده که نتایج حاصله در این مقاله ارائه شده است.

کلمات کلیدی: حل مدل‌های کارایی، زنجیره‌های مارکوف زمان‌پیوسته، الگوریتم‌های موازی، پردازنده‌های چندهسته‌ای.

۱- مقدمه

مدل‌سازی ساخت یک نمونه انتزاعی از سیستم است، به‌صورتی که قابلیت مطالعه و تحلیل آن فراهم شود. مدل‌سازی یکی از گام‌های تحلیل، طراحی و ساخت سیستم‌ها است و با فنون تحلیلی، شبیه‌سازی و تلفیقی امکان‌پذیر است. فنون تحلیلی (مبتنی بر روش‌های ریاضی و صوری) یا منجر به راه‌حل‌های شکل‌بسته[۱] شده، یا با روش‌های عددی قابل حل هستند. هدف حل مدل، ارزیابی کمّی آن و به‌دست آوردن معیارهای کارایی[۲] و اتکاپذیری[۳] است.

زنجیره‌های مارکوف زمان‌پیوسته[۴] (CTMCs)، به‌طور گسترده برای مدل‌سازی و تحلیل کارایی و اتکاپذیری سیستم‌ها مورد استفاده قرار می‌گیرند. با محاسبه احتمالات حالات پایدار CTMC، بسیاری از معیارهای مفید کارایی به دست خواهند آمد. اما مدل‌های CTMC برای سیستم‌های واقعی، بسیار بزرگ بوده و از مشکل انفجار فضای حالت رنج می‌برند.

تاکنون فنون متعددی برای حل تحلیلی[۵] مدل‌های مارکوفی ارائه شده است. فنون موجود سعی نموده‌اند به طرق مختلف مشکلات و سربارهای حل تحلیلی مدل‌ها را، که شامل حافظه و زمان است کاهش دهند و بتوانند مدل‌های بزرگتری را در زمان کمتر، با دقت بالاتری حل کنند. راه‌حل‌هایی نیز برای غلبه بر مشکلات مذکور ارائه شده‌اند که به‌صورت موازی یا توزیع‌شده، از توان محاسباتی بیشتری بهره می‌برند. در [۱] و [۲] از روش‌های صریح[۶] ذخیره‌سازی در حافظه اصلی کامپیوترهای موازی استفاده شده است. برای موازی‌سازی همچنین از روش‌های ضمنی[۷] نیز می‌توان استفاده نمود. در [۳] و [۴] از روش مبتنی بر کرونکر[۸]، و در [۵] و [۶] از روش مبتنی بر نمودار تصمیم‌گیری دودویی چندپایانه‌ای[۹] (MTBDD) استفاده شده است.

از آنجایی‌که روش‌های ضمنی، از یک نمایش ضمنی و فشرده شده برای حل مدل‌های بزرگ استفاده می‌کنند، در این مقاله قصد داریم تا با استفاده از روش‌های ضمنی یک پیاده‌سازی موازی جدید، برای حل حالت پایدار مدل‌های CTMC ارائه

دهیم. برای این منظور از MTBDD برای ذخیره‌سازی CTMC استفاده می‌کنیم، زیرا مبتنی بر استخراج قواعد و ساختار مدل‌ها است و یک نمایش فشرده از سیستم‌های بزرگ را فراهم می‌کند [۷]. همچنین، با استفاده از ترکیب روش‌های مستقیم و تکراری حل دستگاه معادلات، یک روش موازی جدید برای حل حالت پایدار CTMC ارائه می‌دهیم. موازی‌سازی هم درسطح نرم‌افزار (با استفاده از روش رنگ‌آمیزی پیشنهادی) و هم به صورت سخت‌افزاری (روی پردازنده‌های چندهسته‌ای) پیاده‌سازی شده است. نتایج تجربی روی مدل‌های محک، آزمایش شده و ضمن مقایسه با نتایج کارهای قبلی، از نظر معیارهای کارایی مانند تسریع[۱۰] و بازدهی[۱۱] مورد ارزیابی قرار گرفته‌اند.

ساختار ادامه این مقاله به صورت زیر است: در بخش دوم مروری بر زنجیره‌های مارکوف خواهیم داشت و برخی از خواص و ویژگی‌های آنها را مطرح می‌کنیم و به بررسی چگونگی حل دستگاه معادلات حاصل از آنها می‌پردازیم. در بخش سوم، راجع به روش‌های حل مسائل عددی صحبت می‌کنیم و روش‌های مستقیم و تکراری حل دستگاه معادلات خطی را بررسی می‌کنیم. در بخش چهارم، چالش‌های موجود در ذخیره‌سازی و حل بهینه CTMC را معرفی کرده و روش‌های ذخیره‌سازی صریح و ضمنی ماتریس‌های خلوت را به اختصار بیان می‌کنیم. در بخش پنجم، کارهای انجام شده برای حل CTMC را مرور کرده و در بخش ششم به معرفی روش پیشنهادی خود برای حل دستگاه معادلات، با دقت بیشتر و سرعت بالاتر می‌پردازیم و روش موازی‌سازی به کار رفته را در سطح نرم‌افزار و سخت‌افزار بیان می‌نماییم. در بخش هفتم نتایج عملی پیاده‌سازی مدل ساخته شده را روی محک‌های مختلف نشان داده و نتایج را با کارهای قبلی مقایسه می‌کنیم. در انتهای این بخش به نتایج ارزیابی کارایی ابزار ساخته شده می‌پردازیم. در خاتمه، در بخش هشتم به نتیجه‌گیری و کارهای آینده می‌پردازیم.

۲- زنجیره‌های مارکوف زمان پیوسته و حل حالت پایدار آنها

در زنجیره‌های مارکوف زمان‌پیوسته، زمان گذر از هر حالت به حالت دیگر طبق یک متغیر تصادفی پیوسته است. زنجیره‌های مارکوف بی‌حافظه هستند، یعنی با دانستن وضعیت کنونی، وضعیت‌های گذشته هیچ تأثیری در وضعیت آینده سیستم ندارند. از آنجا که تنها یک متغیر تصادفی پیوسته وجود دارد که دارای خاصیت بی‌حافظه بودن است و آن متغیر تصادفی نمایی است، پس در CTMC زمان گذر از هر حالت به حالت دیگر دارای توزیع نمایی است. درواقع، یک CTMC با استفاده از یک توزیع اولیه $\pi(0)$ و یک ماتریس نرخ- گذر- حالت Q به‌صورت فرمول (۱) قابل توصیف است:

$$Q_{ij} = \begin{cases} \lambda_{ij}, & i \neq j \\ -\sum_{j \neq i} Q_{ij}, & i = j \end{cases} \qquad (۱)$$

که در آن λ_{ij} نرخ گذر از حالت i به حالت j است. در حالت پایدار با استفاده از معادلات جریان، دستگاه معادلات $\pi Q = 0$ بدست می‌آید که در آن π بیانگر وضعیت سیستم در حالت پایدار است.

هدف، حلّ دستگاه معادلات $\pi Q = 0$ است تا بردار π بدست آید و فرض بر این است که CTMC کاهش‌ناپذیر است. دستگاه معادلات فوق را می‌توان به فرم $Q^t \pi^t = 0$ نیز نوشت. در این‌صورت هر عنصر قطری در هر ستون ماتریس Q^t برابر منفی مجموع تمام عناصر در همان ستون خواهد بود و همواره هر سطر از این ماتریس را می‌توان از جمع‌کردن تمام سطرهای دیگر بدست آورد. پس همواره هر سطر، ترکیبی خطّی از سایر سطرهای ماتریس است. در نتیجه ماتریس فوق از مرتبه n نخواهد بود، بلکه با یک ماتریس منفرد مواجه هستیم که فاقد جواب یکتاست. این ماتریس از مرتبه $n-1$ است[۸، ۹]. از آنجایی‌که $\sum_{i=1}^{n} \pi_i = 1$ است، می‌توانیم با جای‌گذاری این معادله به‌جای یکی از سطرهای ماتریس Q^t ماتریسی از مرتبه n به دست آوریم. واضح است که این چنین جای‌گذاری که همراه با حذف یکی از سطرهای ماتریس است، دانش مسئله را کم نخواهد کرد. زیرا معادله مربوط به سطر حذف شده همواره به صورت ترکیب خطّی از سایر سطرهای ماتریس است و هر جوابی را که در سطرهای دیگر صدق کند، خواهد پذیرفت. اما، با افزودن این معادله جدید، بردار سمت راست، دیگر بردار صفر نخواهد بود، بلکه درایه متناظر با معادله جدید در آن بردار، مقدار یک خواهد داشت. اینک، یک دستگاه غیر همگن از مرتبه n داریم که نامنفرد بوده و دارای جواب یکتاست. معادله جدید را می‌توان به‌جای هر سطر دلخواهی از ماتریس قرار داد ولی معمولاً آن‌را به‌جای سطر آخر قرار می‌دهند. این ماتریس قطر غالب[۱۲] است زیرا اعضای قطری آن بزرگتر از سایر اعضا در آن ستون هستند. این موضوع یک مزیت به هنگام حل ماتریس با یک روش مستقیم به منظور کاهش خطاهای عددی محسوب می‌شود.

۳- روش‌های حل عددی دستگاه معادلات خطی

دستگاه معادلات رابطه (۲) را در نظر بگیرید:

$$Ax = b,\ x \in R^n \qquad (۲)$$

جواب معادله فوق، $x = A^{-1}b$ است. این حل دقیق معمولاً به دلیل وجود خطاهای عددی، غیرممکن است. اما می‌توان از الگوریتم‌های عددی برای تقریب زدن راه حل استفاده کرد. برای حل این معادله دو دسته از الگوریتم‌ها وجود دارند: الگوریتم‌های مستقیم[۱۳] و الگوریتم‌های تکراری[۱۴]. در ادامه این بخش، به بررسی مختصر این دو دسته از الگوریتم‌های حل دستگاه معادلات خطی خواهیم پرداخت.

۳-۱- روش‌های مستقیم حلّ دستگاه معادلات خطّی

اگر فرض کنیم هیچ خطای گرد کردنی وجود ندارد، روش‌های مستقیم پس از تعداد معینی از مراحل به جواب می‌رسند. این روش‌ها گاهی اوقات روش‌های دقیق نیز نامیده می‌شوند [۱۰]. روش‌های مستقیم با توجه به ابعاد ماتریس، برای حلّ دستگاه به زمان اجرای مشخصی نیاز دارند. اما با افزایش ابعاد ماتریس، تعداد اعمال محاسباتی به سرعت افزایش می‌یابند. پردازش روی ماتریس‌های مورد مطالعه در این مقاله، از چند جهت هزینه دارد: اول آنکه به دلیل خلوت بودن ماتریس‌ها و بزرگی ابعاد آنها، می‌بایست تنها عناصر غیرصفر را در مکان‌های متوالی از حافظه ذخیره نمود. اما، پردازش ماتریس، بعضی از عناصر صفر را که در فایل ذخیره نشده‌اند به عناصری غیرصفر تبدیل می‌کند. این پدیده را پُر شدن[۱۵] می‌نامند [۱۰]. درج این عناصر جدید در فایل به گونه‌ای که همانند بقیه عناصر قابلیت دسترسی سریع به آنها وجود داشته باشد، مسئله‌ای چالش‌برانگیز است. همچنین گاهی اوقات چنین پردازشی منجر به این خواهد شد که ماتریس کلاً خاصیت خلوت بودن خود را از دست بدهد و به یک ماتریس متراکم[۱۶] تبدیل شود [۱۱] . در این‌صورت ممکن است دیگر فضای لازم برای ذخیره‌سازی چنین ماتریس‌هایی وجود نداشته باشد.

روش‌های مستقیم برای حل دستگاههای بزرگ (بیش از هزار حالت) کارایی ندارند، ولی برای دستگاه‌های کوچک می‌توان از روش‌های مستقیم استفاده کرد [۱۱]. با صرف‌نظر کردن از خطاهای عددی که توسط ماشین‌ها بوجود می‌آیند، روش‌های مستقیم تضمین می‌نمایند که با انجام تعداد مشخصی از عملیات به جواب دقیق می‌رسند. این موضوع عمده‌ترین مزیت روش‌های مستقیم نسبت به روش‌های تکراری است.

معروف‌ترین روش مستقیم حل دستگاه، روش حذفی‌گاوس[۱۷] است، (این الگوریتم به عنوان یک روش مستقیم مبنایی محسوب می‌شود و معمولاً سایر الگوریتم‌ها با این الگوریتم مقایسه می‌شوند) [۱۱]. سایر روش‌ها از قبیل روش فاکتورگیری چولسکی[۱۸] این محدودیت را دارند که ماتریس ضرایب دستگاه باید مثبت و متقارن باشد یا برای حل دستگاه‌هایی با ماتریس ضرایب متراکم مناسب هستند. برای مطالعه سایر روش‌های مستقیم از قبیل روش گاوس-جردن به [۲]، فاکتورگیری چولسکی به [۱۳] و فاکتورگیری WZ به [۱۴] مراجعه فرمایید.

از آنجا که ماتریس به دست آمده از CTMC نامنفرد است و بردار سمت راست b مخالف صفر است، با اعمال الگوریتم حذفی گاوس به صورت پسرو، می‌توانیم جواب منحصر بفرد دستگاه را بدست آوریم. در الگوریتم پیشنهادی در ادامه این مقاله از روش مستقیم حذفی گاوس برای حل دستگاه معادلات استفاده می‌شود. الگوریتم محاسبه جواب منحصر به فرد دستگاه در روش حذفی گاوس به صورت زیر است [۲۶]:

۱) $i=1$

۲) اگر $a_{ii}\neq 0$ به مرحله ۴ می‌رویم و اگر $a_{ii}=0$ باشد به مرحله ۳ می‌رویم.

۳) با این فرض که $i<p\leq n$ کوچکترین عدد صحیحی باشد که $a_{pi}\neq 0$، اگر چنین p ای موجود باشد، عمل $E_p \leftrightarrow E_i$ (یعنی تعویض سطر i ام و p ام) را انجام داده به مرحله ۴ می‌رویم و اگر چنین p ای موجود نباشد دستگاه فاقد جواب منحصر بفرد است و الگوریتم متوقّف می‌شود.

۴) به ازای $j=i+1, i+2, ..., n$ ، $m_{ji}=a_{ji}/a_{ii}$ را محاسبه کرده و اعمال $E_j \leftarrow (E_j - m_{ji}E_i)$ را انجام می‌دهیم.

۵) $i=i+1$

۶) اگر $i<n$ به مرحله ۲ می‌رویم و اگر $i=n$ به مرحله ۷ می‌رویم.

۷) اگر $a_{nn}=0$ دستگاه فاقد جواب منحصربفرد است و الگوریتم متوقّف می‌شود و اگر $a_{nn}\neq 0$ ابتدا $X_n=a_{n,n+1}/a_{n,n}$ را بدست می‌آوریم سپس

$$X_i=\frac{(b_i-\sum_{j=i+1}^{n}a_{ij}X_j)}{a_{ii}}$$

را به ازای $i=n-1,...,1$ محاسبه می‌کنیم.

الگوریتم فوق با اعمال سطری پلکانی نظیر $E_j \leftarrow (E_j - m_{ji}E_i)$، ماتریس را به یک ماتریس بالامثلثی تبدیل می‌کند. سپس با جای‌گذاری پسرو، مقادیر x_i را پیدا می‌کند.

۳-۲- روش‌های تکراری حلّ دستگاه معادلات خطّی

روش‌های تکراری، فنونی برای ارائه یک راه حل تقریبی برای دستگاه‌های معادلات خطی هستند. این روش‌ها با یک بردار جواب تقریبی اولیه آغاز به کار کرده و در طی تکرارهای مختلف، بردار جواب را بهبود می‌بخشند تا تقریب‌های به دست آمده به مقادیر واقعی جواب نزدیکتر شوند. این روند تکراری تا زمانی ادامه می‌یابد که بردار جواب با دقت دلخواه به دست آید و نتایج تقریبی به نتایج واقعی بردار جواب همگرا شوند. دو نوع عمده روش‌های تکراری عبارتند از: روش‌های تکراری ایستا[۱۹] که از لحاظ پیاده‌سازی ساده هستند اما معمولاً دقت آنها پایین‌تر است و روش‌های تکراری پویا (نایستا[۲۰])، که تحلیل آنها کمی سخت‌تر است اما دقت و کارایی بالاتری دارند [۱۵].

روش‌های تکراری برای حلّ دستگاه‌های خطّی مزایایی دارند، از جمله: عدم نیاز به پردازش ماتریس ورودی، استفاده محدود از حافظه، سرعت بیشتر نسبت به روش‌های مستقیم، پیاده‌سازی ساده‌تر و امکان همروندسازی بیشتر [۱۳]، اما دقت آنها کمتر از دقت روش‌های مستقیم است. از آنجا که روش‌های مستقیم به دلیل نیاز به پردازش ماتریس ورودی، نیاز به حافظه بیشتری دارند و برای حل دستگاه‌های معادلاتی که ماتریس ضرایب آنها بزرگ است مناسب نیستند، پس برای ماتریس مورد بررسی در این مقاله (ماتریس به دست آمده از فضای حالت زنجیره‌های مارکوف)، استفاده از روش‌های تکراری حل دستگاه معادلات بر روش‌های مستقیم، برتری خواهند داشت. معروف‌ترین الگوریتم‌های تکراری ایستا، الگوریتم‌های ژاکوبی[۲۱] و گاوس- سایدل[۲۲] هستند.

روش ژاکوبی، اگرچه درجه توازی بالایی دارد اما سرعت همگرایی آن نسبتاً پایین است. روش گاوس- سایدل برخلاف روش ژاکوبی، به این دلیل که در هر مرحله از آخرین مقادیر تقریبی به دست آمده برای بردار x استفاده می‌نماید سرعت همگرایی بالاتری دارد. یکی دیگر از محاسن روش گاوس- سایدل این است که می‌توان آن را تنها با استفاده از یک بردار تکراری پیاده‌سازی کرد که این موضوع وقتی که دستگاه معادلات خطی بزرگ باشد اهمیت پیدا می‌کند. با این وجود استفاده از مقادیر تقریبی به دست آمده در هرمرحله، درجه توازی این روش را پایین می‌آورد و آنرا ذاتاً ترتیبی می‌سازد [۱۶].

روش محاسبه دنباله تقریب‌های جواب دستگاه معادلات خطی $Ax=b$، در روش گاوس‌سایدل در فرمول (۳) نشان داده شده است:

$$X_i^{(k)}=\frac{b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k-1)}}{a_{ii}} \qquad (۳)$$

که b_i عنصر iام از بردار جواب b و a_{ii} عنصر قطری در سطر iام و $X_i^{(k)}$ مقدار عنصر iام از بردار جواب در دستگاه معادلات خطی است. توجه کنید که در گام iام از الگوریتم، iامین عنصر x با استفاده از مقادیر فعلی سایر عناصر بردار به روزرسانی می‌شود. به همین دلیل است که تنها نیاز به نگهداری یک بردار جواب داریم و سرعت همگرایی نیز بیشتر است [۶].

۴- انگیزش

۴-۱- روش‌های موازی حل تکراری دستگاه معادلات خطی خلوت بزرگ

حل دستگاه معادلات خطی خلوت، قلب بسیاری از محاسبات علمی است. زیرا بسیاری از مسائل علمی و مهندسی، یک دستگاه معادلات خطی خلوت تولید می‌کنند. یکی از این قبیل مسائل، زنجیره‌های مارکوف زمان‌پیوسته هستند که در بخش ۲ به معرفی آنها پرداختیم. CTMCها، برای تحلیل کارایی سیستم‌های کامپیوتری و ارتباطی مورد استفاده قرار می‌گیرند. با محاسبه احتمالات حالت پایدار CTMC، بسیاری از مقادیر مفید کارایی به دست خواهند آمد. اما مدل‌های CTMC برای سیستم‌های واقعی بسیار بزرگ هستند و از مشکل انفجار فضای حالت رنج می‌برند [۱۶]. معمولاً یک سیستم درحالت کلی، متشکل از چندین زیرسیستم است که هریک از آنها وظیفه خاصی را به انجام می‌رسانند و اندازه

فضای حالت کلی سیستم معمولاً نمایی است. مشکل انفجار فضای حالت با استفاده از روش‌های نمادین[۲۳] مرتفع می‌گردد.

دو روش عمده برای ذخیره‌سازی ماتریس‌های خلوت عبارتند از روش‌های صریح و روش‌های ضمنی. می‌توان حل CTMC را به صورت حل یک دستگاه معادلات خطی خلوت به فرم $Ax=b$ ساده‌سازی کرد که اندازه دستگاه برابر با تعداد حالات CTMC است.

۴-۲- روش ذخیره‌سازی صریح ماتریس‌های خلوت

یک ماتریس خلوت، ماتریسی است که اعضای غیرصفر بسیار کمی دارد. اما در عمل، یک ماتریس زمانی خلوت نامیده می‌شود که بتوان فنون خاصی را که از مزیت تعداد زیاد اعضای صفر و محل آنها در ماتریس استفاده می‌کنند، روی ماتریس به‌کار برد [۱۱].

به منظور بهره‌گیری از مزیت صفرهای زیاد، ماتریس‌های خلوت باید به روش‌های خاصی ذخیره شوند. هدف اصلی این است که تنها عناصر غیر صفر ماتریس نمایش داده شوند و اینکه قادر به انجام عملیات‌های ماتریسی معمول روی آنها باشیم. یک روش استاندارد برای حل حالت پایدار CTMCها استفاده از روش‌های صریح است، روش‌هایی که فضای حالت و ساختمان داده مربوط به CTMC را با استفاده از فنون ذخیره‌سازی خلوت ذخیره می‌کنند. روش‌های صریح معمولاً به خاطر سرعت و ساختار مبتنی بر آرایه، راه حل سریعتری را ارائه می‌کنند. اگرچه این روش‌ها تنها می‌توانند مدل‌هایی را حل کنند که داخل حافظه قرار داشته باشند [۱۶]. در ادامه این بخش به معرفی برخی از مهم‌ترین روش‌های ذخیره‌سازی صریح ماتریس‌های خلوت می‌پردازیم. از نماد N_Z برای نشان دادن تعداد کلّ عناصر غیر صفر استفاده می‌کنیم.

۴-۲-۱- قالب مختصاتی

ساده‌ترین روش ذخیره‌سازی صریح ماتریس‌های خلوت، استفاده از قالب مختصاتی است. ساختمان داده در این روش شامل سه آرایه است:

۱) یک آرایه حقیقی شامل تمام مقادیر حقیقی غیرصفر ماتریس A با هر ترتیب دلخواه

۲) یک آرایه صحیح جهت نگهداری اندیس‌های سطری در آرایه قبلی

۳) یک آرایه صحیح شامل اندیس‌های ستونی همان آرایه. هر یک از این سه آرایه به طول N_Z (تعداد متغیرهای غیرصفر) هستند.

۴-۲-۲- قالب سطرهای خلوت فشرده

قالب سطرهای خلوت فشرده[۲۴] (CSR) معروف‌ترین و پرطرفدارترین روش ذخیره‌سازی ماتریس‌های خلوت است. ساختمان داده آن، از سه آرایه برای ذخیره‌سازی ماتریس خلوت با ابعاد $n \times n$ استفاده می‌کند [۱۷]:

۱) یک آرایه حقیقی *Value* شامل مقادیر حقیقی و غیرصفر a_{ij} که به‌صورت سطر به سطر ذخیره شده‌اند. ماتریس شامل a مقدار غیرصفر غیرقطری و n مقدار غیرصفر قطری است. بنابراین طول آرایه *Value*، $N_Z = n + a$ است.

۲) یک آرایه مقدار صحیح *Column* شامل اندیس‌های ستونی مقادیر a_{ij}ای که در آرایه *Value* ذخیره شده‌اند. طول آرایه *Column*، $N_Z = n + a$ است.

۳) یک آرایه مقدار صحیح *Start* شامل اشاره‌گرهایی به شروع هر سطر در آرایه *Value* و *Column*. بنابراین مقدار $Start(i)$، مکانی در آرایه‌های *Value* و *Column* است که i امین سطر شروع می‌شود. طول آرایه *Start* برابر $n+1$ است که مقدار $Start(n+1)$ برابر با $Start(1) + N_Z$ است.

۴-۲-۳- قالب سطرهای خلوت تغییریافته

از آنجا که اکثر روش‌های حل دستگاه معادلات به صورت ویژه‌ای با عناصر قطری ماتریس سروکار دارند، پس بهتر است که عناصر قطری ماتریس در یک آرایه اعشاری به طول n ذخیره شوند. این کار باعث می‌شود که نیاز به ذخیره‌سازی اندیس ستون‌ها را برای مقادیر قطری نداشته باشیم. این شمای ذخیره‌سازی با قالب جدیدی با نام سطرهای خلوت تغییریافته[۲۵] (*MSR*) معروف است که همان *CSR* تغییریافته است [۱۷].

قالب *MSR* تنها به دو آرایه نیاز دارد: یک آرایه حقیقی *Value* و یک آرایه صحیح *Column*. n مکان نخست در آرایه *Value* شامل عناصر قطری ماتریس اصلی بصورت مرتّب است. مکان $n+1$ام آرایه *Value* مورد استفاده قرار نمی‌گیرد اما می‌تواند جهت نمایش اطّلاعات دیگری از ماتریس مورد استفاده قرار گیرد. با شروع از مکان $n+2$، عناصر غیر صفر *Value* با صرف‌نظر از عناصر قطری آن به ترتیب سطری ذخیره می‌شوند. به ازای هر عنصر $Value(k)$، عدد صحیح $Column(k)$ بیانگر شماره ستون آن عنصر در ماتریس است. اما $n+1$ مکان نخست در *Column* شامل اشاره‌گری به شروع هر سطر در آرایه‌های *Value* و *Column* است.

۴-۳- روش ضمنی ذخیره‌سازی ماتریس‌های خلوت

دسته دیگری از روش‌ها برای ذخیره‌سازی ماتریس‌های خلوت، روش‌های ضمنی هستند. این روش‌ها مبتنی بر استخراج قواعد و ساختار مدل‌ها هستند و بنابراین می‌توانند یک نمایش فشرده از سیستم‌های بزرگ را ارائه دهند. اما یک محدودیت در روش‌های ضمنی، نیاز به ذخیره‌سازی حتمی بردارهای راه حل است.

این روش‌ها به دلیل استفاده از ساختمان داده مبتنی بر آرایه، راه حل با سرعت بالاتری فراهم می‌نمایند، اگرچه تنها قادر به حل مدل‌هایی هستند که داده‌ها بتوانند همزمان با هم در تمام ایستگاه‌های کاری در داخل حافظه‌اصلی (RAM) قرار بگیرند. در مقایسه با روش‌های درون هسته‌ای[۲۶]، روش‌هایی که برون هسته‌ای[۲۷] نامیده می‌شوند از حافظه دیسک برای غلبه بر محدودیت‌های حافظه استفاده می‌کنند و بهبود چشم‌گیری در اندازه مدل‌های قابل حل روی یک ماشین ایجاد نموده‌اند [۱۶].

۴-۳-۱- نمایش نمادین و ساختار بلاکی

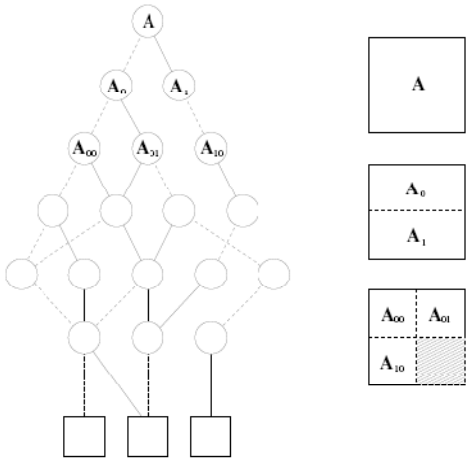
نمایش نمادین مدل‌های احتمالی چندین برابر کوچکتر از آلترناتیوهای صریح از قبیل ماتریس‌های خلوت است. علاوه بر مزیت فشردگی نمایش مدل، مزیت دیگر مدل‌های نمادین امکان ساخت مدل به صورت سریع و مناسب است. برای این منظور ساختمان داده‌های مختلفی معرفی شده‌اند. یک انتخاب مناسب، استفاده از MTBDD برای ذخیره‌سازی فضای حالت است. زیرا اولاً تا حد ممکن ارتباطات داخل یک پردازنده را کاهش می‌دهد و ثانیاً به راحتی اجازه دسترسی به اطلاعات مربوط به وابستگی وظایف را می‌دهد [۱۸].

روش‌های ضمنی به کمک جبر خطی، فضای حالت مربوط به CTMC را با استفاده از تکنیک‌های ذخیره‌سازی خلوت (نمودار تصمیم‌گیری دودویی چندپایانه‌ای) ذخیره می‌کنند. این روش‌ها دارای مزایای زیر هستند: اولاً نمایش فشرده‌تری از مدل فراهم می‌کنند و سرعتشان به اندازه سرعت حل ماتریس‌های

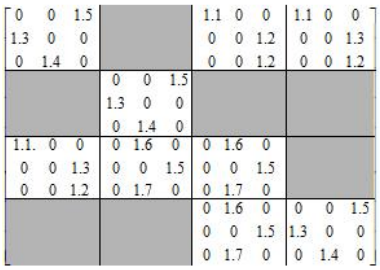
خلوت است. ثانیاً، با وجود اینکه نمادین هستند، درجه بالایی از توازی را فراهم می‌کنند [۵].

MTBDD، یک درخت دودویی کاهش یافته است که یک تابع با مقادیر حقیقی را با استفاده از متغیرهای بولی نمایش می‌دهد. ماتریس نرخ گذر CTMC تابعی است که زوج حالت‌ها را به مقادیر حقیقی نگاشت می‌کند. برای تعیین احتمال یا نرخ هر زوج حالت، باید مسیر را از بالا به پایین درخت پیگیری کرد و در هرگره از درخت با توجه به مقادیر بولی تصمیم‌گیری انجام داد [۵].

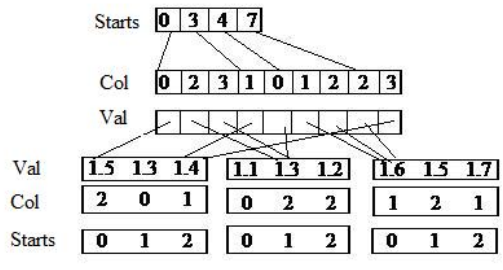
شکل (۱) ساختار یک MTBDD را برای ماتریس دلخواه A نشان می‌دهد. هر گره MTBDD یک بلاک از ماتریس A است و هر دو یال خارج شونده از آن، تجزیه آن را به دو زیرماتریس دیگر نشان می‌دهد. ماتریس A گره ریشه است و با یک حرکت در سطح پایین‌تر، دو گره فرزند نشان‌گر تقسیم‌بندی A به دو زیرماتریس A_0, A_1 هستند. با یک حرکت به سطح سوم هرکدام از گره‌های A_0, A_1 به دو گره چپ و راست تقسیم می‌شوند. در این مثال چهارمین گره سمت راست خالی است و بنابراین دومین یال خارج شونده از A_1 نشان داده نشده است. شکل (۲) نمایش MTBDD را برای ماتریس یک CTMC نشان می‌دهد.



شکل ۱- نمایش ماتریس A به فرم MTBDD و تجزیه آن به زیرماتریس‌ها [۶]



الف- ماتریس یک CTMC



ب- نمایش به فرم یک MTBDD تغییر یافته

شکل ۲- ماتریس یک CTMC و نمایش آن به فرم MTBDD تغییر یافته [۱۷]

می‌توان عناصر قطری ماتریس CTMC را به صورت مجزا در یک آرایه ذخیره کرد. عناصر قطری ماتریس در شکل (۲) (الف)، همگی صفر هستند. ماتریس به 4^2 بلاک تقسیم می‌شود که بعضی از بلاک‌ها صفر هستند. اطلاعات بلاک‌های غیرصفر در ماتریس تنها یک بار با قالب CSR با استفاده از سه آرایه Starts و Col و Val ذخیره می‌شوند (قسمت پایین شکل (۲) (ب) را ببینید).

۵- مروری بر کارهای مرتبط

از آنجا که الگوریتم گاوس- سایدل ذاتاً ترتیبی بوده و عملیات موازی‌سازی آن مشکل است، پیاده‌سازی موازی آن برمبنای کاربردهای خاص انجام گرفته است. در این بخش مروری برکارهای انجام شده برای حل دستگاه معادلات به دست آمده از زنجیره‌های مارکوف خواهیم داشت. لازم به ذکر است که تمامی این کارها، برروی معماری سیستم‌های چند پردازنده‌ای پیاده‌سازی شده‌اند، اما از آنجا که بهره‌گیری از فناوری پردازنده‌های چندهسته‌ای طی سالهای اخیر حتی در سطح رایانه‌های شخصی نیز رواج یافته است، در این مقاله، الگوریتم پیشنهادی برای حل دستگاه معادلات CTMC بر روی معماری چندهسته‌ای ارائه خواهد شد. در کارهایی که تاکنون انجام شده است، از ایده رنگ‌آمیزی گراف یا فن جبهه‌موج[۲۸] برای بهبود موازی‌سازی بهره گرفته شده است. این روش‌ها ترتیب اجرای محاسبات را در هر تکرار تغییر می‌دهند تا وظایفی را که به یکدیگر وابستگی ندارند همزمان با هم انجام دهند و سرعت انجام محاسبات کلی افزایش یابد.

در [۱۹] حل معادلات دیفرانسیل جزئی و در [۲۰] حل دستگاه معادلات متراکم سیستم‌های توان الکتریکی با استفاده از روش گاوس- سایدل موازی انجام گرفته است. در [۲۴] حل دستگاه معادلات خطی خلوت با استفاده از الگوریتم GMRES توصیف شده است. در [۷] راجع به روش ژاکوبی موازی بحث شده است و به کمک آن یک روش موازی برای حل حالت پایدار CTMC ارائه شده است. در [۶] فضای حالت CTMC با استفاده از MTBDD ذخیره شده است و موازی‌سازی روش گاوس- سایدل با کمک فن جبهه‌موج انجام گرفته است.

در [۱۷] یک راه حل موازی ضمنی برای حل زنجیره‌های مارکوف با استفاده از روش ژاکوبی آورده شده است. در [۲۵] یک روش جدید حل CTMC برمبنای ترکیب دو روش تکراری ژاکوبی و گاوس- سایدل ارائه شده است. زیرا روش ژاکوبی ذاتاً ماهیت موازی دارد و برای اجرا روی کامپیوترهای خوشه‌ای[۲۹] مناسب است و روش گاوس‌سایدل هم با وجود آنکه ماهیت ترتیبی دارد اما درجه همگرایی آن بالاتر از روش ژاکوبی است.

فن جبهه‌موج، دستگاه معادلات را به یک سری جبهه‌موج تقسیم‌بندی می‌نماید. مجهولات در داخل هر جبهه‌موج می‌توانند به صورت غیرهمگام[۳۰] در داخل پردازنده‌های مختلف محاسبه شوند. اما فن رنگ‌آمیزی درجه توازی بالاتری به دست خواهد داد. در این فن، گره‌های یک گراف به گونه‌ای رنگ‌آمیزی می‌شوند که هیچ دو گره مجاوری هم‌رنگ نباشند. هدف به دست‌آوردن حداقل تعداد رنگ‌های ممکن است. این کار یک عملیات پیچیده با زمان محاسبات نمایی است. از این رو، در این مقاله یک روش رنگ‌آمیزی ساده، برای به دست آوردن گراف وابستگی وظایف در CTMC پیشنهاد خواهیم نمود.

۶- روش پیشنهادی برای حل دستگاه معادلات

روش‌های تکراری برای حل دستگاه‌هایی کاربرد دارند که فضای حالت بسیار بزرگی دارند. لذا ما برای محاسبه احتمالات حالت پایدار در حالت کلی از روش تکراری گاوس- سایدل استفاده می‌کنیم. اگرچه الگوریتم گاوس- سایدل ذاتاً ترتیبی است اما از آنجایی که ماتریس به دست آمده از مدل CTMC خلوت است می‌توان توازی

را از الگوی بلاک‌های غیرصفر ماتریس استخراج کرد [۶].

در واقع می‌توان طبیعت ترتیبی الگوریتم گاوس- سایدل را با فرمول‌بندی بلاک‌ها تخفیف داد. فرض کنید ماتریس A به $N\times N$ بلاک با اندازه $m\times m$ تقسیم‌بندی شده باشد و بلاک (p,q)ام با $A_{(p,q)}$ نمایش داده شود و بردارها نیز به زیربردارهایی با اندازه متناسب تقسیم‌بندی گردند. p امین بلاک بردار x با x_p نمایش داده می‌شود. (i,j)امین عنصر از زیرماتریس $A_{(p,q)}$ و i امین عنصر از زیربردار x_p به ترتیب با $A_{(p,q)i,j}$ و $x_{(p)i}$ نمایش داده می‌شوند. اندازه بلاک ماتریس $A_{(p,q)}$، $n_p\times n_q$ است و اندازه x_p نیز n_p است.

در هر تکرار، محاسبه p امین بلاک x_p بستگی به مقادیر ورودی‌ها در سایر بلاک‌های x دارد. در واقع بستگی به تمام بلاک‌های $x_{(q)}$ دارد که در آنها $A_{(pq)}$ خالی نباشد. خوشبختانه ماتریس بدست آمده از حل CTMC خلوت است و تعداد زیادی بلاک خالی در آن وجود دارد.

با استفاده از این نمادگذاری، نسخه مبتنی بر بلاک یک تکرار از الگوریتم گاوس- سایدل در شکل (۳) نمایش داده شده است. گام p ام از حلقه بیرونی مقادیر جدید عناصر x_p از بردار حل را محاسبه می‌کند. این کار با افزودن یک ماتریس کوچک v با اندازه $max_{0\le p<N}\{n_p\}$ انجام می‌گیرد. مهم‌ترین جنبه این الگوریتم این است که در هر زمان به جای دسترسی به یک عنصر، می‌توان به یک بلاک از عناصر دسترسی پیدا کرد.

1. $for\,(0\le p<N)$
2. $\quad v:=b_{(p)}$
3. $\quad foreach\ block\ A_{(pq)}\ with\ q\neq p$
4. $\quad\quad v:=v-A_{(pq)}X_{(q)}$
5. $\quad for(0\le i<n_p, i\neq j)$
6. $\quad\quad X_{(p)i}:=(v_i-\sum_{0\le j<n_p} A_{(pp)ij}.X_{(p)j})\Big/A_{(pp)ii}$

شکل ۳- الگوریتم تکراری گاوس- سایدل به فرم بلاک‌بندی شده [۶]

از نوشتن تکرارهای الگوریتم گاوس- سایدل روی بلاک‌ها، چندین دستگاه معادلات با اندازه‌های کوچک‌تر در داخل بلاک‌ها حاصل می‌شود. واضح است که اندازه این بلاک‌ها به مراتب از اندازه ماتریس ضرایب اولیه کوچکتر است. در [۱۷] حل دستگاه‌های معادلات به دست آمده از داخل بلاک‌ها به روش تکراری ژاکوبی و در [۱۹] به روش تکراری گاوس- سایدل انجام گرفته است. اما پیشنهاد ما استفاده از یک روش مستقیم برای حل دستگاه‌های معادلاتی است که از داخل بلاک‌ها حاصل شده‌اند، زیرا:

۱) روش‌های تکراری به ندرت برای حلّ دستگاه‌های خطّی با اندازه کوچک به‌کار می‌روند، چون زمان لازم برای حصول دقّت کافی در این دستگاه‌ها، بیش از زمان لازم برای روش‌های مستقیم نظیر روش حذفی گاوس برای ماتریس‌هایی با این اندازه است.

۲) دقت روش‌های مستقیم برای محاسبه بردار جواب از دقت روش‌های تکراری بیشتر است.

اما برای انتخاب روش حل عددی مستقیم مناسب با فرضیات مسئله، باید توجه داشت که ماتریس‌های به دست آمده از داخل بلاک‌ها نیز خلوت هستند. از بین روش‌های مستقیم حل دستگاه، به دلایلی که در زیر مطرح می‌شود روش حذفی گاوس مناسب‌تر از سایر روش‌ها به نظر می‌رسد:

- نسبت به سایر روش‌های مستقیم سرعت بیشتری داشته و به تعداد اعمال محاسباتی کمتری نیاز دارد.
- از آنجا که تنها تعداد کمی از درایه‌های زیر قطر اصلی در هربلاک مقدار غیرصفر دارند، مسئله متراکم شدن چالش‌برانگیز نخواهد بود.
- در ماتریس به دست آمده از *CTMC*، مشکل محورگیری در روش حذفی گاوس وجود ندارد، زیرا مقادیر روی قطر اصلی به مراتب بزرگتر از مقدار سایر درایه‌ها در همان ستون از ماتریس هستند.

۶-۱- استخراج اطلاعات وابستگی‌ها

الگوریتم به N گام تقسیم‌بندی شده است که هرگام یک بلاک از بردار جواب x را محاسبه می‌کند. این گام‌ها به صورت موازی بین هسته‌های پردازنده تقسیم می‌شوند.

برای ذخیره‌سازی ماتریس ضرایب A، که یک ماتریس مربعی است و اندازه آن $n\times n$ است، مقادیر درایه‌های غیرصفر ماتریس به همراه اندیس سطر و ستون آنها به عنوان ورودی‌های مسئله داده می‌شوند. سپس برمبنای اندازه ماتریس، عملیات بلاک‌بندی درایه‌ها صورت می‌گیرد. بدین ترتیب که ماتریس ضرایب خلوت مسئله، به یک سری بلاک‌های کوچکتر با اندازه $m\times m$ تقسیم‌بندی می‌شود (لزومی ندارد که اندازه تمام بلاک‌ها با یکدیگر یکسان باشد، اما در اینجا برای سادگی کار، فرض می‌کنیم اندازه همه بلاک‌ها یکسان است). یک سری از این بلاک‌ها شامل مقادیر غیرصفر هستند و یک سری از این بلاک‌ها صرفاً شامل درایه‌هایی با مقدار عددی صفر هستند. از ذخیره‌سازی بلاک‌هایی که تمام درایه‌های آنها مقدار صفر دارند صرف‌نظر کرده و در سایر بلاک‌ها نیز از روش ذخیره‌سازی خلوت CSR استفاده می‌نماییم، یعنی تنها اندیس سطر و ستون و مقدار درایه غیرصفر داخل حافظه نگه‌داری می‌شود تا از مصرف بیش از حد حافظه و اشباع و انبار کردن تعداد زیادی مقدار غیرلازم جلوگیری شود. به این ترتیب ویژگی محلیت داده‌ها هم حفظ می‌شود، یعنی وقتی تنها مقادیر غیرصفر ماتریس را ذخیره‌سازی می‌کنیم، مشکل انفجار فضای حالت حل می‌شود و چون ماتریس ذاتاً خلوت است و تعداد مقادیر غیرصفر آن زیاد نیستند، پس این مقادیر می‌توانند همزمان در حافظه محلی تمام هسته‌های پردازنده قرار گیرند و عملیات رفت و برگشت به حافظه تکرار نشود. عملیات اصلی حل دستگاه روی بلاک‌ها انجام می‌گیرد.

۶-۲- رنگ‌آمیزی گراف وظایف

یکی از عوارض موازی‌سازی روش تکراری گاوس- سایدل این است که هنگام محاسبه بلاک X_j از بردار جواب، باید برای تمام بلاک‌های X_i که $i<j$ است از مقادیر به روزرسانی شده استفاده کنیم (مقادیری که در همین تکرار محاسبه شده‌اند) و این همان چیزی است که الگوریتم گاوس- سایدل را ذاتاً ترتیبی می‌سازد.

اگرچه می‌توان به صورت امنی ترتیب اجرا را به هم ریخت و محاسبه بلاک‌های X_i را در یک ترتیب متفاوت به انجام رسانید. این کار جایگشت‌های مختلف ترتیب عناصر در ماتریس A و بردارهای b و x در دستگاه معادلات خطی $Ax=b$ است، که هیچ تاثیری روی حل آن ندارد. می‌توانیم از این واقعیت استفاده کنیم و یک دنباله‌ای از ترتیب‌های اجرا را پیدا کنیم که قابلیت موازی‌سازی داشته باشد. در واقع، اگر بتوانیم تعیین کنیم کدام بلاک‌ها مستقل از یکدیگر هستند آنگاه می‌توانیم در هر زمان به جای محاسبه یک بلاک از بردار جواب، چندین بلاک را به صورت موازی با یکدیگر محاسبه نماییم.

اما، برای اینکه بدانیم کدام دنباله از X_i ها را می‌توانیم به صورت موازی با هم محاسبه کنیم، لازم است بدانیم که بردار جواب X_j به کدامیک از بردارهای جواب X_i (برای $j>i$) وابسته است.

یک روش به دست آوردن وابستگی‌ها در محاسبه بردارهای جواب، استفاده از ایده رنگ‌آمیزی گراف‌ها است. برای این منظور می‌بایست گراف وابستگی وظایف (بلاک‌ها) به یکدیگر را ترسیم نمود. اگر مقدار $A_{i,j}$ در گراف وابستگی مخالف صفر باشد، آنگاه یک یال از رأس i به رأس j ترسیم خواهد شد که نشان‌دهنده وابستگی محاسباتی بردار جواب X_j به بردار X_i است. واضح است که اگر بین دو رأس یالی ترسیم شده باشد آنگاه این دو رأس نمی‌توانند همرنگ باشند. پس اگر مقدار بردار X_j به بردار X_i (برای $i<j$) وابستگی داشته باشد، آنگاه تا زمانی که بردار X_i محاسبه نشده باشد، نمی‌توان بردار X_j را محاسبه کرد. اما اگر در ماتریس ضرایب، بلاک $A_{i,j}$ مقدار صفر داشته باشد، یعنی هیچ وابستگی داده‌ای بین بردارهای جواب X_j و X_i وجود نداشته باشد، می‌توان این دو بردار جواب را به صورت همزمان و موازی با هم محاسبه نمود.

در اینجا نشان خواهیم داد یک الگوریتم رنگ‌آمیزی گراف اگرچه دنباله‌ای از ترتیب‌های اجرای موازی بردارهای جواب را خواهد داد، اما برای مسئله ما بهینه نیست و با تغییر الگوریتم رنگ‌آمیزی گراف، آنرا خاص مسئله خود می‌سازیم. ویژگی‌های الگوریتم پیشنهادی ما به شرح زیر است:

۱) اگرچه این الگوریتم از ایده رنگ‌آمیزی گراف استفاده می‌کند، اما پیاده‌سازی آن تفاوت‌هایی با نحوه پیاده‌سازی الگوریتم‌های رنگ‌آمیزی گراف دارد. در واقع در اینجا، درجه هر رأس از گراف اهیمت دارد (یعنی یک رأس به چه تعدادی از رئوس وابسته است). برای مثال اگر یک بلاک از ماتریس به دو بلاک دیگر وابسته باشد، آنگاه درجه رأس متناظر با آن بلاک در گراف ۲ خواهد بود. تا زمانی‌که، تمام بلاک‌هایی که در بردار جواب برای یک رأس ایجاد وابستگی کرده‌اند محاسبه نشده باشند، درجه رأس متناظر با آن بلاک در گراف صفر نخواهد شد و این بلاک از بردار جواب به دلیل وجود وابستگی‌ها موقتاً قابل محاسبه نیست.

۲) این الگوریتم پیشنهاد شده، نسبت به الگوریتم‌های رنگ‌آمیزی گراف ساده‌تر بوده و پیچیدگی محاسباتی آن بسیار کمتر است و دنباله‌ای از ترتیب‌های اجرا را فراهم خواهد نمود که کامل‌ترین و بهینه‌ترین حالت ممکن است.

درادامه با یک مثال الگوریتم رنگ‌آمیزی درحالت معمول را روی یک گراف اعمال می‌کنیم و ضمن بیان ضعف آن برای بدست آوردن دنباله وابستگی‌ها در ماتریس حالت، الگوریتم پیشنهادی را که متناسب با ساختار CTMC است، ارائه خواهیم داد.

مثال: در شکل (۴) دنباله‌ای از رئوس گراف و یال‌هایی که وابستگی این رئوس به یکدیگر را مشخص می‌کنند داده شده‌اند. در اینجا اندیس i ($0\le i<11$) شماره هر رأس را مشخص می‌سازد.

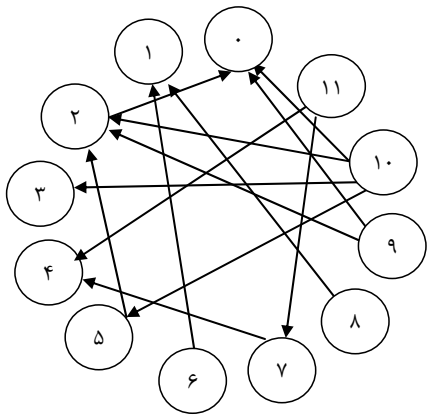
جدول ۱- رنگ‌آمیزی گراف وابستگی وظایف در شکل (۴)

رنگ	شماره رأس
۱	۶ و ۸ و ۹ و ۱۰ و ۱۱
۲	۳ و ۵ و ۷
۳	۱ و ۲ و ۴
۴	۰

با استفاده از الگوریتم رنگ‌آمیزی گراف، به رئوسی که به یکدیگر وابستگی ندارند و می‌توانند مستقل از هم اجرا شوند، رنگ یکسانی اختصاصی می‌دهیم. برای مثال دو رأس ۰ و ۲ که مجاور هستند (یالی بین آنها ترسیم شده است و به یکدیگر وابستگی دارند) نمی‌توانند همرنگ باشند و به صورت موازی با هم اجرا شوند. پس ابتدا باید رأس ۰، و سپس رأس ۲ که به آن وابسته است، پردازش گردد. جدول (۱) شماره رنگ هر رأس را در گراف وابستگی وظایف مشخص می‌سازد. همانطور که گفتیم این رنگ‌آمیزی بهینه نیست. به طور مثال، طبق این الگوریتم رئوس ۱ و ۲ و ۴ که همرنگ هستند می‌توانند همزمان با هم اجرا شوند. اما همانطور که در گراف ملاحظه می‌شود رأس صفر که به هیچ رأس دیگری وابسته نیست و می‌تواند همزمان با رئوسی ۱ و ۳ و ۴ اجرا شود، شماره رنگ یکسان با این رئوس ندارد. پس الگوریتم رنگ‌آمیزی گراف علاوه بر تحمیل پیچیدگی زمانی و حافظه مصرفی زیاد به برنامه، کارایی لازم را هم نخواهد داشت.

۶-۳- الگوریتم پیشنهادی برای رنگ‌آمیزی گراف وابستگی بین وظایف

ابتدا تمام وظایف را داخل مجموعه‌ای به نام *unprocessed* قرار می‌دهیم. عملیات محاسبه هر بردار جواب x_p را یک وظیفه t_p می‌نامیم و برای هر تکرار الگوریتم گاوس- سایدل داریم : $T=\{t_p: 0\le p<N\}$. مجموعه تمام وظایفی که وظیفه t_p به آنها وابسته است با D_p نشان می‌دهیم به طوریکه: $D_p=\{t_q\in T: q\neq p \text{ and } A_{(p,q)} \text{ is non-empty}\}$. می‌توان یک گراف وابستگی وظایف $G=(T,E)$ ایجاد کرد به طوری‌که رئوس T وظایفی هستند که در مجموعه *unprocessed* قرار دارند و یالهای $E\subseteq T\times T$ نشان‌دهنده وابستگی وظایف به یکدیگر هستند، به طوریکه $(t_p,t_q)\in E$ است، اگر وتنها اگر $t_q\in D_p$ باشد. از نماد *Color* برای نشان دادن تعداد کل رنگ‌های به کار رفته در گراف استفاده می‌کنیم. هرچه مقدار *Color* کوچکتر باشد، درجه توازی به دست آمده برای مسئله بزرگتر خواهد بود.



شکل ۴- گراف وابستگی وظایف بردارهای جواب

الگوریتم پیشنهادی به صورت زیر است:

۱) $Color=1$.

۲) تا زمانی که مجموعه *unprocessed* حداقل یک عضو دارد مراحل ۳ تا ۵ را به صورت تکراری انجام دهید:

۳) از مجموعه *unprocessed*، تمام وظایف t_q را که هیچ وابستگی به سایر وظایف ندارند و مجموعه D_q آنها *null* است (درجه رأس آنها صفر است) خارج کنید و آنها را با رنگ *Color* شماره‌گذاری نمایید. اگر مجموعه *unprocessed* تهی شد به مرحله ۶ بروید.

۴) $Color=Color+1$

۵) درجه هر رأس t_p را که به مجموعه‌ای از رئوس t_q وابسته است (و آن رئوس قبلاً پردازش شده‌اند و به آنها شماره رنگ اختصاص داده شده است و از مجموعه *unprocessed* حذف شده است)، یک واحد کاهش دهید.

۶) پایان.

جدول (۲) نتایج رنگ‌آمیزی با الگوریتم پیشنهادی را نشان می‌دهد.

جدول ۲- رنگ‌آمیزی گراف وابستگی وظایف با الگوریتم پیشنهادی

رنگ	شماره رأس
۱	۰ و ۱ و ۳ و ۴
۲	۲ و ۶ و ۷ و ۸
۳	۵ و ۹ و ۱۱
۴	۱۰

۶-۴- تحلیل الگوریتم پیشنهادی

اگر هر یک از رئوس را معادل یک بردار جواب X_i از بردار حل در نظر بگیریم، آنگاه مطابق جدول (۲) برای محاسبه بردارهای جواب X_0, X_1, X_3, X_4 نیازی به انتظار برای محاسبه هیچ بردار جواب دیگری نداریم، زیرا این بردارهای جواب به هیچ بردار دیگری وابسته نیستند. پس این بردارها را از مجموعه بردارهای پردازش نشده حذف می‌کنیم و به آنها رنگ ۱ را اختصاص می‌دهیم و آنها را به صورت موازی با هم محاسبه می‌کنیم. در مرحله بعد می‌توانیم بردارهای جواب X_2, X_6, X_7, X_8 را که به بردارهای جواب X_0, X_1, X_3, X_4 وابسته هستند، موازی با هم محاسبه نماییم و از مجموعه بردارهای پردازش نشده حذف کنیم. این کار تا جایی ادامه می‌یابد که تمام بردارهای جواب محاسبه شوند.

پس از توصیف روش پیشنهادی برای ساخت حل‌کننده مدل‌های کارایی، اینک به بررسی روش پیاده‌سازی آن روی پردازنده‌های چندهسته‌ای می‌پردازیم.

۶-۵- پیاده‌سازی با استفاده از استخر وظایف

برای پیاده‌سازی ابزار حل‌کننده موازی مدل‌های کارایی، از فن چندنخی[۳۱] در زبان برنامه نویسی C#، چارچوب .NET و محیط Visual Studio 2010 استفاده کرده‌ایم.

یک نخ معمولاً باید تعداد زیادی از وظایف را اجرا نماید. یک ساختار ساده می‌تواند به این صورت باشد که هر وظیفه‌ای در یک تابع جداگانه قرار داده شود و بعداً توسط یک نخ صدا زده شده و اجرا گردد. وابسته به دانه‌بندی وظایف، این روش منجر به تولید تعداد زیادی نخ می‌شود که سربار زیادی دارد. اما یک ایده بهتر استفاده از *استخر وظایف*[۳۲] است. پیاده‌سازی این روش به این صورت است که وظایف آماده اجرا داخل یک صف قرار گیرند و برای اجرا تعداد مشخصی نخ که توسط نخ اصلی[۳۳] تولید شده‌اند در نظر گرفته شوند. نخ‌ها برای بازیابی وظایف از صف آماده اجرا، فراخوانی شده و وظایف را یکی پس از دیگری به انجام برسانند. در حین اجرای وظایف، اگر وظیفه جدیدی تولید شود به انتهای صف آماده اجرا افزوده خواهد شد [۱۸].

مزیت این روش این است که با ثابت نگه داشتن تعداد نخ‌ها، هرقدر هم که وظایف آماده اجرا زیاد باشند مدیریت نخ‌ها ساده بوده و مستقل از تعداد وظایف بوده و سربار تولید نخ نخواهیم داشت.

با تقسیم وظایف محاسباتی که در پردازنده‌های تک‌هسته‌ای اجرا می‌شدند و محول کردن هر بخش از کار به یک هسته در پردازنده‌های چندهسته‌ای، تعداد کارهای بیشتری را می‌توان در همان سیکل ساعت انجام داد. برای این منظور، نرم‌افزار مورد اجرا باید به گونه‌ای نوشته شده باشد که بتواند بار کاری را برای اجرا روی هسته‌های مختلف پخش کند. برای تعیین حداکثر و حداقل تعداد نخ‌هایی که بایستی به صورت موازی با هم، به محاسبه بلاک‌های بردار جواب CTMC بپردازند، از کد شکل (۵) استفاده می‌شود. این کد بیان می‌کند که برای نمونه، اگر بخواهیم برنامه را روی دو هسته از پردازنده اجرا کنیم، باید دو نخ به صورت موازی با هم کار کنند.

```
using System.Threading;
ThreadPool.SetMinThreads(2, 0);
ThreadPool.SetMaxThreads(2, 0);
```

شکل ۵- تعیین تعداد نخ‌ها برای اجرای برنامه به صورت موازی

همانطور که در شکل (۶) ملاحظه می‌شود، با توجه به الگوریتم رنگ‌آمیزی پیشنهادی، ابتدا یک واحد از درجه وابستگی هر بلاک به بلاک‌هایی که قبلاً محاسبه شده‌اند کاسته می‌شود، اگر درجه وابستگی صفر شد آن بلاک از بردار جواب آماده محاسبه شدن است و داخل صف آماده اجرا قرار می‌گیرد. هر نخ پس از اتمام محاسبات خود منتظر اتمام کار نخ‌های دیگر می‌ماند تا عملیات همگام‌سازی انجام گیرد.

```
if (Interlocked.Decrement(ref  DependencyNumber == 0)
    ThreadPool.UnsafeQueueUserWorkItem(WaitCallBack
callback ,object state);
completedEvent.WaitOne();
```

شکل ۶- قرار گرفتن وظایف مستقل داخل صف آماده اجرا و انتظار برای اتمام کار نخ‌های دیگر

۷- نتایج ارزیابی روش پیشنهادی

در این بخش نتایج ارزیابی حاصل از پیاده‌سازی ابزار حل‌کننده موازی مدل‌های کارایی ارائه گردیده است. برای آزمایش مدل از یک پردازنده چهار هسته‌ای با مشخصات سخت‌افزاری زیر:

Intel®Core™ i7 CPU 930@2.8 GH, 3 GB RAM

که به خاطر بهره‌گیری از فناوری ابرنخی[۳۴] قادر است تا هشت نخ را به صورت موازی با هم اجرا کند و سه مدل محک CTMC استفاده شده است؛ سیستم ساخت و تولید انعطاف‌پذیر[۳۵](FMS) [22]، سیستم ساخت و تولید کَن‌بَن[۳۶][۲۳] و سیستم سرکشی سرویس‌دهنده چرخشی[۳۷] [۲۴].

برای آزمایش هر محک، از ماتریس‌ها با اندازه‌های مختلف استفاده شده و احتمالات حالات پایدار را محاسبه نموده‌ایم. محاسبات در جایی خاتمه می‌یابد که اختلاف مقادیر به دست آمده در بردارهای حل دو تکرار کمتر از 10^{-6} باشد، یعنی زمانی‌که رابطه (۴) برقرار گردد:

$$\max_{0\le i<n}\left(\frac{|X_i-\tilde{X}_i|}{|X_i|}\right)<10^{-6} \qquad (۴)$$

که X و $\tilde{X}$ به ترتیب مقادیر بردارهای حل در تکرار فعلی و تکرار قبلی هستند.

۷-۱- ارزیابی کارایی

یک معیار مهم در ارزیابی کارایی، زمان پاسخ[۳۸] برنامه (فاصله زمانی بین شروع و اتمام اجرای برنامه) است. ویژگی‌های ماتریس‌های مورد استفاده در هر محک و زمان اجرای هر محک به روش پیشنهادی بر روی پردازنده با هسته‌های مختلف در جدول‌های (۳)، (۴) و (۵) آمده است. همانطور که در این جدول‌ها ملاحظه می‌فرمایید زمان اجرای پیاده‌سازی موازی روی معماری چندهسته‌ای بسیار کمتر

از زمان اجرا به صورت ترتیبی روی یک هسته پردازنده است. همچنین زمان اجرای روش پیشنهادی در این مقاله نسبت به روش‌هایی که در کارهای پیشین روی معماری‌های چند پردازنده‌ای پیاده‌سازی شده بودند، کاهش بسیار چشم‌گیری داشته است. شکل‌های (۷)، (۸) و (۹) به مقایسه زمان اجرای روش پیشنهادی برای حل مدل‌های CTMC (ترکیب روش مستقیم حذفی‌گاوس و روش تکراری گاوس‌سایدل)، با روش انجام شده در [۶] (حل دستگاه به روش تکراری گاوس‌سایدل) می‌پردازد.

بر طبق قانون آمدال[۳۹]، با افزایش تعداد هسته‌ها کارایی تا یک جایی افزایش خواهد یافت و بعد از آن هر چقدر تعداد هسته‌ها را بیشتر کنیم کارایی ثابت باقی می‌ماند [۲]. در واقع این قانون بیان می‌کند که تسریع در اجرای برنامه تابعی از کسری از برنامه است که می‌تواند به صورت پرشتاب اجرا شود. بنابراین، حداکثر تسریع، به بخشی از برنامه که به صورت ترتیبی اجرا می‌شود وابسته است و مستقل از تعداد پردازنده‌ها است [۱۲]. پس کاهش بخشی از کد که به صورت ترتیبی اجرا می‌شود و افزایش بخش موازی، مهم‌تر از افزودن تعداد هسته‌های بیشتر است و بیشتر روی کارایی اثر دارد. در ادامه این بخش به بررسی معیارهای تسریع و بازدهی روش پیشنهادی برای حل مدل‌های CTMC می‌پردازیم.

۷-۲- تسریع

برای تحلیل برنامه‌های موازی، باید زمان اجرای پیاده‌سازی ترتیبی برنامه با حالت اجرای موازی مقایسه شود. یک چنین مقایسه‌ای مرتبط با زمان اجرای کم شده در حالت موازی است و تسریع نام دارد. تسریع $S_p(n)$ در یک برنامه موازی با زمان اجرای $T_p(n)$ به صورت رابطه (۵) تعریف می‌شود [۱۸]:

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \qquad (۵)$$

که p تعداد پردازنده‌ها برای حل مسئله‌ای با ابعاد n است. $T^*(n)$، بهترین زمان پیاده‌سازی ترتیبی برنامه، و $T_p(n)$ زمان اجرای موازی برنامه روی p پردازنده است. تسریع در اجرای یک برنامه موازی، به مقدار زمانی برمی‌گردد که می‌تواند در مقایسه با زمان اجرای ترتیبی همان برنامه کاهش داده شود. شکل (۱۰)، (۱۱) و (۱۲) میزان تسریع به دست آمده از اجرای روش پیشنهادی را روی محک‌های مختلف نشان می‌دهد. همان‌طور که در شکل‌های (۱۰) الی (۱۲) ملاحظه می‌فرمایید، با افزایش اندازه ماتریس مسئله، میزان تسریع نیز افزایش می‌یابد، بنابراین می‌توانیم انتظار داشته باشیم که هرچه اندازه ماتریس‌های مسئله را افزایش دهیم، میزان تسریع نیز افزایش خواهد یافت.

۷-۳- بازدهی

طبق تعریف، بازدهی کسری از زمان است که پردازنده به صورت مفید و بدون درنظر گرفتن سربار، مشغول انجام محاسباتی هست که در حالت ترتیبی هم در برنامه وجود دارند. تعریف بازدهی برمبنای هزینه برنامه موازی است و می‌تواند به صورت رابطه (۶) بیان شود [۱۸]:

$$E_p(n)=\frac{T^*(n)}{C_p(n)}=\frac{S_p(n)}{p}=\frac{T^*(n)}{p.T_p(n)} \qquad (۶)$$

که $T^*(n)$ زمان اجرای برنامه در حالت ترتیبی و $T_p(n)$ زمان اجرای موازی برنامه روی p پردازنده است. اگر هیچ تسریع خطی رخ نداده باشد آنگاه $E_p(n)\geq 1$.

تسریع ایده آل $S_p(n)=p$ است که معادل با $E_p(n)=1$ است. شکل (۱۳)، (۱۴) و (۱۵) بازدهی به دست آمده از اجرای روش پیشنهادی را روی محک‌های مختلف نشان می‌دهد.

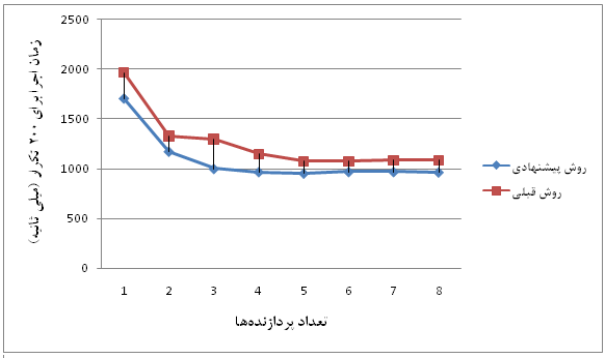
۷-۴- مقیاس‌پذیری

مقیاس‌پذیری[۴۰] برنامه‌های موازی، معیاری است که تعیین می‌کند که با افزایش تعداد پردازنده‌ها، کارایی به چه میزان افزایش پیدا می‌کند [۱۸]. اگر با ثابت نگه داشتن اندازه مسئله (n)، تعداد پردازنده‌ها افزایش یابد و یا با ثابت نگه‌داشتن تعداد پردازنده‌ها ابعاد مسئله کاهش پیدا نماید، آنگاه کارایی سیستم افزایش می‌یابد. مقیاس‌پذیری پارامتر مهمی در برنامه‌نویسی موازی است زیرا می‌توان مسائل با ابعاد بزرگ را در زمانی معادل با زمان موردنیاز برای مسائل با ابعاد کوچکتر حل کرد به شرط آنکه از تعداد بیشتری پردازنده استفاده شود. همانطور که در جدول‌های (۳)، (۴) و (۵) ملاحظه می‌فرمایید روند کاهش زمان اجرا از پنج هسته به بعد یک روند ثابت و خطی بوده و افزایش هسته تاثیر زیادی در کاهش زمان اجرا ندارد.

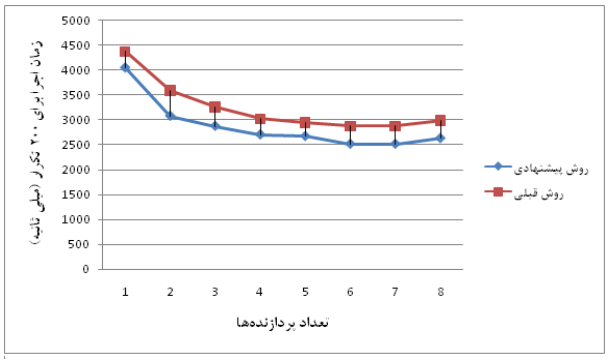
۸- نتیجه‌گیری

با استفاده از روش‌های نمادین، حل حالت پایدار CTMC با فضای حالت بزرگ و خلوت را به صورت موازی روی پردازنده‌های چندهسته‌ای پیاده‌سازی کردیم. برای حل دستگاه معادلات خطی بزرگ و خلوت CTMC یک روش جدید (استفاده ترکیبی از روش‌های تکراری برای بلاک‌ها و روش مستقیم برای حل دستگاه به دست آمده از داخل بلاک‌ها) ارائه نمودیم که سرعت و دقت حل مدل را افزایش می‌دهد. موازی‌سازی هم درسطح نرم‌افزار (با استفاده از روش رنگ‌آمیزی پیشنهادی) و هم به صورت سخت‌افزاری (روی پردازنده‌های چندهسته‌ای) پیاده‌سازی گردیده و با مدل‌های محک آزمایش شده و مورد ارزیابی قرار گرفته‌اند.

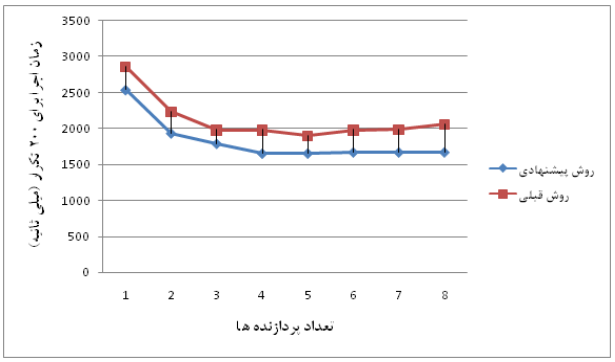
روش پیشنهادشده در این مقاله بر روی معماری چندهسته‌ای پیاده‌سازی و آزمایش شده است. در یک تحقیق دیگر می‌توان به پیاده‌سازی آن مبتنی بر بهره‌گیری ترکیبی از پردازنده چندهسته‌ای و واحد پردازش گرافیکی[۴۱] (GPU) پرداخت. ضمناً از آنجا که روش‌های تکراری برای حل دستگاه‌های بزرگ مناسب هستند، در این پایان‌نامه از الگوریتم تکراری گاوس- سایدل که یک روش تکراری ایستا است بهره گرفتیم. اما برای حل دستگاه معادلات خطی بزرگ می‌توان از روش‌های تکراری پویا، نظیر روش گرادیان مزدوج[۴۲]، روش BiCG[۴۳]، روش QMR[۴۴] و غیره نیز استفاده کرد، که هرچند از نظر تحلیلی سخت‌تر هستند، ولی دارای کارایی بالاتری هستند.



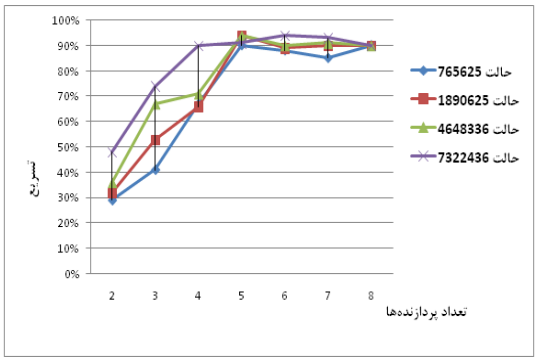
شکل ۷- مقایسه زمان اجرای روش پیشنهادی برای آزمایش محک ساخت و تولید انعطاف‌پذیر با کارهای قبلی بر روی پردازنده با هسته‌های مختلف



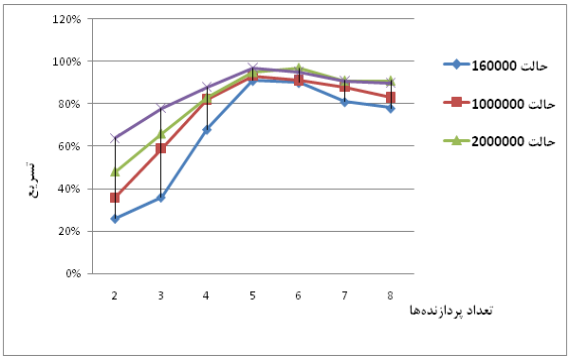
شکل ۸- مقایسه زمان اجرای روش پیشنهادی برای آزمایش محک کَن‌بَن با کارهای قبلی بر روی پردازنده با هسته‌های مختلف



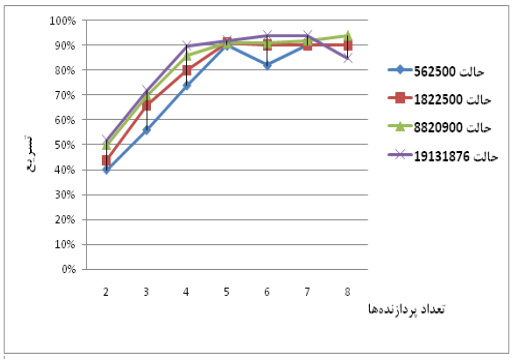
شکل ۹- مقایسه زمان اجرای روش پیشنهادی برای آزمایش محک سیستم سرکشی سرویس‌دهنده چرخشی با کارهای قبلی بر روی پردازنده با هسته‌های مختلف



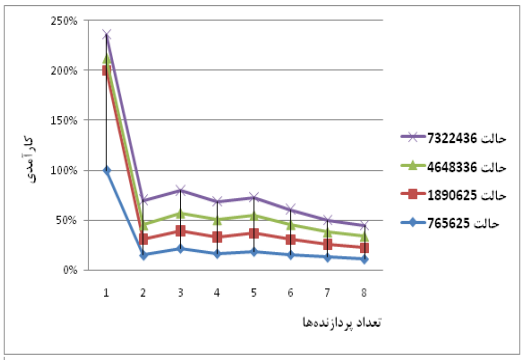
شکل ۱۰- تسریع به‌دست آمده از اجرای روش پیشنهادی روی محک ساخت و تولید انعطاف‌پذیر با اندازه‌های مختلف ماتریس



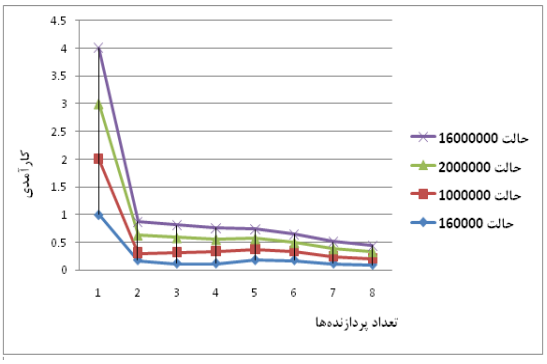
شکل ۱۱- تسریع به‌دست آمده از اجرای روش پیشنهادی روی محک کَن‌بَن با اندازه‌های مختلف ماتریس‌ها



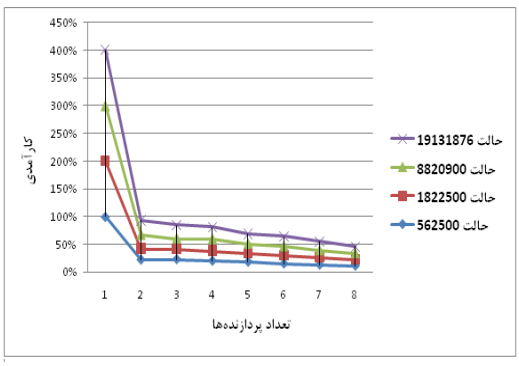
شکل ۱۲- تسریع به‌دست آمده از اجرای روش پیشنهادی روی محک سیستم سرکشی سرویس‌دهنده چرخشی با اندازه‌های مختلف ماتریس



شکل ۱۳- بازدهی به‌دست آمده از اجرای روش پیشنهادی روی محک ساخت و تولید انعطاف‌پذیر با اندازه‌های مختلف ماتریس



شکل ۱۴- بازدهی به‌دست آمده از روش پیشنهادی روی محک کَن‌بَن با اندازه‌های مختلف ماتریس‌ها



شکل ۱۵- بازدهی به‌دست آمده از اجرای روش پیشنهادی روی محک سیستم سرکشی سرویس‌دهنده چرخشی با اندازه‌های مختلف ماتریس

جدول ۳- فضای حالت و زمان اجرای محک ساخت و تولید انعطاف‌پذیر بر روی پردازنده چندهسته‌ای

اندازه فضای حالت	تعداد و اندازه بلاک‌ها		زمان اجرای ۲۰۰ تکرار بر روی هسته‌های پردازنده (میلی ثانیه)						
		تعداد هسته‌ها:	۱	۲	۳	۴	۵	۶	۷	۸
۷۶۵۶۲۵	$۱۷۵^۲\times۵^۲$	روش پیشنهادی	۳۷۶	۳۲۸	۲۸۱	۲۸۱	۲۵۰	۲۵۰	۲۵۵	۲۵۸
		روش قبلی	۴۱۱	۳۴۱	۳۱۲	۲۸۸	۲۸۰	۲۸۰	۲۸۱	۲۸۵
۱۸۹۰۶۲۵	$۲۷۵^۲\times۵^۲$	روش پیشنهادی	۶۳۵	۵۴۴	۵۰۲	۴۷۶	۴۳۵	۴۳۸	۴۳۵	۴۳۵
		روش قبلی	۸۱۸	۷۰۱	۶۱۶	۵۷۰	۵۶۵	۵۶۵	۵۶۶	۵۷۱
۴۶۴۸۳۳۶	$۵۳۹^۲\times۴^۲$	روش پیشنهادی	۱۵۳۵	۱۳۶۵	۱۲۷۸	۱۱۳۵	۱۰۳۹	۱۰۴۰	۱۰۳۸	۱۰۳۹
		روش قبلی	۱۸۵۹	۱۴۳۸	۱۲۹۷	۱۲۶۶	۱۲۷۶	۱۲۷۶	۱۳۱۳	۱۳۱۵
۷۳۲۲۴۳۶	$۴۵۱^۲\times۶^۲$	روش پیشنهادی	۱۴۵۵	۱۱۷۲	۱۰۵۹	۱۰۰۲	۱۰۱۱	۱۰۰۲	۱۰۰۵	۱۰۰۵
		روش قبلی	۱۹۰۵	۱۵۲۸	۱۴۵۰	۱۳۸۸	۱۳۱۱	۱۳۱۰	۱۳۱۱	۱۳۱۰

جدول ۴- فضای حالت و زمان اجرای محک کَن‌بَن بر روی پردازنده چندهسته‌ای

اندازه فضای حالت	تعداد و اندازه بلاک‌ها		زمان اجرای ۲۰۰ تکرار بر روی هسته‌های پردازنده (میلی ثانیه)						
		تعداد هسته‌ها:	۱	۲	۳	۴	۵	۶	۷	۸
۱۶×۱۰^۴	$۸۰^۲\times۵^۲$	روش پیشنهادی	۲۰۳	۱۷۲	۱۷۲	۱۶۳	۱۴۰	۱۴۰	۱۵۱	۱۶۲
		روش قبلی	۲۰۳	۱۷۲	۱۷۲	۱۶۳	۱۴۰	۱۵۶	۱۷۲	۱۷۲
$۱۰^۶$	$۲۰۰^۲\times۵^۲$	روش پیشنهادی	۵۷۸	۵۱۱	۴۴۵	۴۰۸	۳۷۵	۳۸۲	۳۸۷	۳۸۵
		روش قبلی	۶۶۰	۵۳۸	۴۷۳	۴۶۵	۴۵۹	۴۵۸	۴۵۸	۴۵۹
۲×۱۰^۶	$۴۰۰^۲\times۵^۲$	روش پیشنهادی	۱۲۶۰	۹۵۳	۹۰۱	۸۷۵	۸۵۰	۸۵۰	۸۵۵	۸۵۵
		روش قبلی	۱۴۳۱	۱۱۰۹	۱۰۶۳	۹۹۷	۹۸۵	۹۸۵	۹۸۵	۱۰۰۰
۱۶×۱۰^۶	$۸۰۰^۲\times۵^۲$	روش پیشنهادی	۳۸۳۰	۳۰۷۸	۲۸۷۵	۲۷۰۴	۲۶۷۲	۲۶۳۰	۲۶۳۰	۲۶۳۰
		روش قبلی	۴۲۱۵	۳۵۹۶	۳۲۶۵	۳۰۳۲	۲۸۸۵	۲۸۸۵	۲۸۸۵	۲۹۵۵

جدول ۵- فضای حالت و زمان اجرای محک سیستم سرکشی سرویس‌دهنده چرخشی بر روی پردازنده چندهسته‌ای

اندازه فضای حالت	تعداد و اندازه بلاک‌ها		زمان اجرای ۲۰۰ تکرار بر روی هسته‌های پردازنده (میلی ثانیه)						
		تعداد هسته‌ها:	۱	۲	۳	۴	۵	۶	۷	۸
۵۶۲۵×۱۰^۲	$۱۵۰^۲\times۵^۲$	روش پیشنهادی	۲۷۳	۲۲۳	۲۰۴	۱۹۷	۱۸۸	۱۸۹	۱۸۹	۱۸۸
		روش قبلی	۲۷۹	۲۳۰	۲۰۹	۱۹۸	۱۹۵	۱۸۹	۱۸۸	۱۸۸
۱۸۲۲۵×۱۰^۲	$۲۷۰^۲\times۵^۲$	روش پیشنهادی	۵۵۴	۴۶۱	۴۱۵	۴۰۲	۳۸۶	۳۷۵	۳۷۶	۳۷۶
		روش قبلی	۵۸۷	۴۶۹	۴۲۳	۳۹۸	۳۸۹	۳۸۶	۳۹۱	۳۹۲
۸۸۲۰۹×۱۰^۲	$۵۹۴^۲\times۵^۲$	روش پیشنهادی	۱۶۸۱	۱۳۴۳	۱۳۱۲	۱۱۷۲	۱۱۷۲	۱۱۷۲	۱۱۷۲	۱۱۹۵
		روش قبلی	۱۸۳۹	۱۴۶۹	۱۴۵۴	۱۳۵۹	۱۳۲۸	۱۳۴۴	۱۳۴۴	۱۳۴۴
۱۹۱۳۱۸۷۶	$۷۲۹^۲\times۶^۲$	روش پیشنهادی	۲۴۴۰	۱۹۳۵	۱۷۹۰	۱۶۵۳	۱۶۵۵	۱۶۷۰	۱۶۷۰	۱۶۷۱
		روش قبلی	۲۸۶۰	۲۲۳۵	۱۹۸۵	۱۹۸۵	۱۹۰۶	۱۹۸۵	۱۹۸۶	۲۰۶۲

مراجع

[1] P. Marenzoni, and S. Caselli, *"Analysis of Large GSPN Models: A Distributed Solution Tool*," Proc. of 7th International Workshop on Petri Nets and Performance Models, pp. 130-143, 1997.

[2] S. C. Allmaier, M. Kowarschik, and G. Horton, *"State space construction and steady-state solution of GSPNs on a shared-memory multiprocessor,"* Proc. of the 6th International Workshop on Petri Nets and Performance Models, IEEE CS Press, pp. 112-121, 1997.

[3] P. Kemper, *"Parallel randomization for large structured Markov chains,"* Proc. of the 2002 International Conference on Dependable Systems and Networks, Washington, DC, USA, pp 657-666, IEEE CS Press, 2002.

[4] P. Buchholz, M. Fischer, and P. Kemper, *"Distributed Steady State Analysis Using Kronecker Algebra"*, Proc. of the 3rd International Workshop on the Numerical Solution of Markov Chains, Zaragoza, Spain, 1999.

[5] M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood, *"Dualprocessor parallelisation of symbolic probabilistic model checking*," Proc. of 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 123–130, 2004.

[6] Y. Zhang, D. Parker, and M. Kwiatkowska, *"A wavefront parallelisation of CTMC solution using MTBDDs*," Proc. of International Conference on Dependable Systems and Networks, IEEE Computer Society Presss, pp. 732–742, 2005.

[7] R. Mehmood, and J. Crowcroft, "Parallel iterative solution method for large sparse linear equation systems," IEEE Transactions on Computers, pp.1554-1568, 2005.

[8] A. Berman, and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, 1979.

[9] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Chichester, West Sussex 1994.

[10] B. Bylina, and J. Bylina, *"A review of numerical methods for solving large Markov chains,"* Proc. ofthe EuroNGI Workshop: New Trends in Modeling, Quantitive Methods and Measurements, Jacek Skalmierski Computer Studio, Gliwice, (2004) 75.

[11] Y. Saad, Iterative Methods for Sparse Linear Systems**,** Second edition, *SIAM*, Philadelphia, 2003.

[12] S. Akhter, and Jason Roberts, *Multi-Core Programming Increasing Performance through Software Multi-Threading*, Intel Press, 2006.

[13] J. Kurzak, and A. Buttari, *"Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization*," IEEE Trans. on Parallel Systems, vol. 19. pp. 1175-1186. 2008.

[14] P. Yalamov, and D. Evans, *"The WZ matrix factorisation method*," Parallel Computing, vol. 21, pp. 1111-1120, 1995.

[15] P. Biswanath, and K. Datta, *Numerical* Linear Algebra *and Applications*, Brooks Publishing Company, New York, 1994.

[16] R. Mehmood, and J. Crowcroft, *"Parallel iterative solution method for large sparse linear equation systems,"* Technical Report UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, 2005.

[17] R. Mehmood, J. Crowcroft, and J. Elmirghani, *"A Parallel Implicit Method for the Steady-State solution of CTMC,"* Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006.

[18] T. Rauber, and G. Runger, *Parallel programming for multicore and cluster systems*, Springer 2010.

[19] G. Golub, J. Ortega, and G. Golub, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, 1993.

[20] D. P. Koester, S. Ranka, and G. C. Fox, *"A Parallel Gauss-Seidel Algorithm for Sparse power system matrices*," Proc. of the 1994 ACM/IEEE conference on Supercomputing, pp. 184-193, 1994.

[21] G. Ciardo, and K. Trivedi, *"A decomposition approach for stochastic reward net models,"* Performance Evaluation, Vol. 18, pp. 37–59, 1993.

[22] G. Ciardo, and M. Tilgner, *on the use of Kronecker operators for the solution of generalized stocastic Petri nets*, ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, 1996.

[23] O. Ibe, and K. Trivedi, *"Stochastic Petri net models of polling systems,"* IEEE Journal on Selected Areas in Communications, Vol. 8, pp. 1649–1657, 1990.

[24] J. Bylina, and B. Bylina, *"A Markovian model of the RED mechanism solved with a cluster of computers,"* Annales UMCS Informatica, vol.5, pp. 19–27, 2006.

[25] J. Bylina, and B. Bylina, *"Merging Jacobi and Gauss-Seidel methods for solving Markov chains on computer clusters*," Proc. of the International Multiconference on Computer Science and Information Technology (IMCSIT), vol. 3, pp. 263-268, 2008.

[۲۶] ر. ا. بوردن، آنالیز عددی، (ترجمه) عالم‌زاده، علی‌اکبر، ۱۳۷۹.

شهرزاد ترابی کارشناسی در رشته مهندسی کامپیوتر- نرم‌افزار را در سال ۱۳۸۶ از دانشکده فنی دکتر شریعتی و کارشناسی ارشد در رشته مهندسی کامپیوتر- نرم‌افزار را در سال ۱۳۹۰ از دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران دریافت نموده است. زمینه‌های تحقیقاتی مورد علاقه ایشان، ارزیابی کارایی سیستم‌های کامپیوتری و طراحی و پیاده‌سازی ابزارهای مدل‌سازی و تحلیل است. از ایشان تاکنون چندین مقاله در کنفرانس‌ها به چاپ رسیده است. خانم ترابی هم‌اکنون در بخش فناوری اطلاعات بانک مرکزی جمهوری اسلامی ایران اشتغال دارند.

آدرس پست‌الکترونیکی ایشان عبارت است از:

sh_torabi@comp.iust.ac.ir

محمد عبداللهی ازگمی کارشناسی، کارشناسی ارشد و دکتری در رشته مهندسی کامپیوتر- نرم‌افزار را به ترتیب در سال‌های ۱۳۷۱، ۱۳۷۵ و ۱۳۸۴ از دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف دریافت نموده است. زمینه‌های تحقیقاتی مورد علاقه ایشان، مدل‌سازی ارزیابی کارایی، اتکاءپذیری و امنیت، طراحی نرم‌افزارهای اتکاءپذیر و امن و درستی‌یابی صوری است. از ایشان تاکنون مقالات متعددی در کنفرانس‌ها و مجلات به چاپ رسیده است. دکتر عبداللهی ازگمی هم‌اکنون استادیار گروه نرم‌افزار در دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران است.

آدرس پست‌الکترونیکی ایشان عبارت است از:

azgomi@iust.ac.ir

اطلاعات بررسی مقاله:

تاریخ ارسال: ۹۰/۲/۲۰

تاریخ اصلاح: ۹۰/۱۲/۱

تاریخ قبول شدن: ۹۰/۱۲/۵

نویسنده مرتبط: دکتر محمد عبداللهی ازگمی، دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران.

[1] Closed-Form Solution
[2] Performance
[3] Dependability
[4] Continuous Time Markov Chains
[5] Analytic Solution
[6] Explicit Methods
[7] Implicit Methods
[8] Kronecker-Based Approach
[9] Multi-Terminal Binary Decision Diagrams
[10] Speedup
[11] Efficiency
[12] Diagonal Dominant
[13] Direct Algorithms
[14] Iterative Algorithms
[15] Fill-In
[16] Dense
[17] Gaussian Elimination
[18] Cholesky Factorization
[19] Stationary
[20] Non-Stationary
[21] Jacobi
[22] Gauss-Seidel
[23] Symbolic
[24] Compressed Sparse Row
[25] Modified Sparse Row
[26] In-Core Approach
[27] Out-Of-Core Approaches
[28] Wavefront Technique
[29] Cluster Computers
[30] Asynchronous
[31] Multi-Threading
[32] Task Pool
[33] Main Thread
[34] Hyper-Threading
[35] Flexible Manufacturing System(Fms)
[36] Kanban Manufacturing System
[37] Cyclic Server Polling System
[38] Response Time
[39] Amdahl’s Law
[40] Scalability
[41] Graphics Processing Unit
[42] Conjugate Gradient
[43] Biconjugate Gradient
[44] Quasi-Minimal Residual