RTHS: یک مرتب‌ساز سخت‌افزاری با کارایی بالا و هزینه کم، با استفاده از یک الگوریتم مرتب‌سازی چندبعدی

پرستو شجاعی سرچشمه، محمد مهدی رجبی، محمدرضا رحیمی جعفری، زهرا سادات موسوی

مرتب سازی انواع مختلفی دارد و بسته به جاهای مختلف و بر اساس نیاز های ما استفاده از هرکدام متفاوت و مطابق با نیاز است، از آنجایی که این مقاله را برای درس FPGA آماده کردیم، ما از یک الگوریتم مرتب سازی چند بعدی برای به وجود آوردن مرتب‌ساز سخت‌افزاری real - time با کارایی بالا و کم هزینه استفاده کردیم و در ادامه به بحث در این مورد میپردازیم. مرتب‌ساز سخت‌افزاری بلادرنگ (RTHS) برای کاربردهای پردازش داده‌های حجیم که در آن عملکرد و صرفه‌جویی در منابع از اهمیت بالایی برخوردار است، استفاده می شود. FPGAها با فراهم آوردن راه‌حل‌هایی با توان عملیاتی بالا و استفاده بهینه از حافظه، امکان پیاده‌سازی طراحی‌های موازی و خطی را برای انواع معماری‌ها فراهم می‌کنند. روش‌های سنتی مانند شبکه مرتب‌سازی بیتونیک (CBSN) علیرغم توان عملیاتی بالا، کارایی حافظه پایینی دارند و نیازمند منابع زیادی برای پیاده‌سازی هستند. اما RTHS با کاهش قابل توجه منابع مورد نیاز، این مشکل را برطرف کرده است.

کلید واژه- شبکه مرتب‌سازی بیتونیک، طراحی با هزینه کم، مرتب‌ساز موازی، مرتب‌ساز زمان واقعی، الگوریتم مرتب‌سازی، شبکه مرتب‌سازی

۱- مقدمه

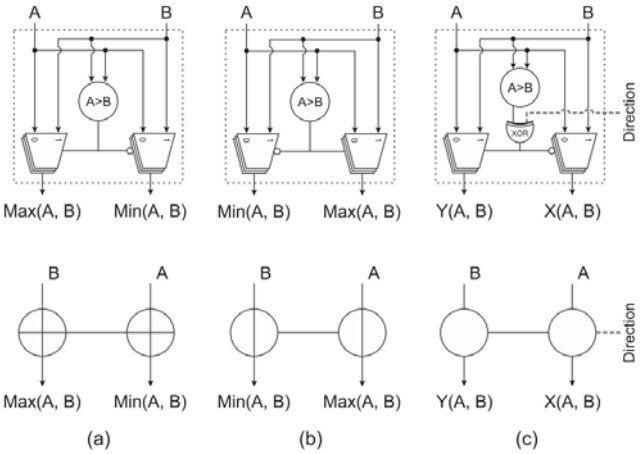
مرتب‌سازی عملیات اساسی در زمینه‌های مختلفی مانند جستجو، پایگاه داده، و هوش مصنوعی است که بر زمان اجرای سیستم‌ها برای پردازش داده‌های بزرگ تأثیر زیادی دارد. الگوریتم‌های مرتب‌سازی نرم‌افزاری نیاز به تعداد زیادی تکرار دارند و با افزایش ورودی‌ها زمان اجرا نیز افزایش می‌یابد. تکنیک‌های جدید با استفاده از پردازنده‌های چند هسته‌ای و GPU به بهبود عملکرد این الگوریتم‌ها کمک کرده‌اند، و اخیراً طراحی شتاب‌دهنده‌های سخت‌افزاری با FPGA مورد توجه قرار گرفته است. طراحی شتاب‌دهنده‌های سخت‌افزاری نیاز به منابع زیادی دارد و زمان اجرای بدترین حالت برای وظایف مرتب‌سازی در سیستم‌های زمان واقعی اهمیت ویژه‌ای دارد. یکی از ابزارهای مفید برای این کار، ساختار صف Min/Max queue است که در کاربردهایی مانند الکترونیک و پزشکی استفاده می‌شود. الگوریتم مرتب‌سازی بیتونیک Batchers با استفاده از معماری سخت‌افزاری موازی برای سریع‌تر کردن عملیات مرتب‌سازی طراحی شده است. این الگوریتم به دلیل نیاز به حافظه زیاد و منابع بالا برای تعداد زیاد ورودی‌ها مشکلاتی دارد، از جمله عدم توانایی در بررسی میانی فرایند مرتب‌سازی و افزایش طول مسیر بحرانی. این مقاله الگوریتم مرتب‌سازی چندبعدی (MDSA) را پیشنهاد می‌کند که به کاهش منابع مورد نیاز، افزایش کارایی حافظه و کاهش تأثیر منفی بر زمان اجرا کمک می‌کند. الگوریتم MDSA به عنوان جایگزینی برای تکنیک‌های دیگر در سیستم‌های زمان واقعی و برای پیاده‌سازی Min/Max queues و یافتن رکوردهای بزرگترین و کوچک‌ترین در داده‌های بزرگ کاربرد دارد. ساختار مقاله به این صورت است: بخش ۲ به معرفی شبکه مرتب‌سازی بیتونیک و بلوک‌های CAS می‌پردازد، بخش ۳ به بررسی سخت‌افزارهای مختلف مرتب‌سازی با FPGA، بخش ۴ الگوریتم MDSA را معرفی می‌کند، بخش ۵ رویکرد پیشنهادی و معماری آن را بررسی می‌کند، و بخش ۶ نتایج تجربی را ارائه می‌دهد و بخش ۷ به جمع‌بندی مقاله می‌پردازد.

۲- پس زمینه

بلوک‌های CAS

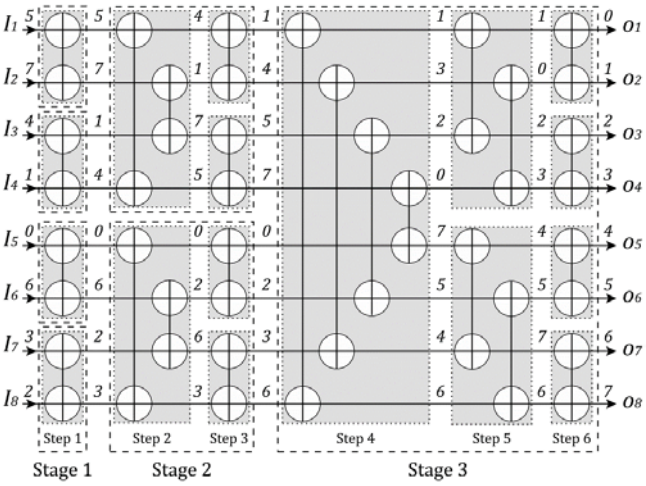
یک شبکه مرتب‌سازی ترکیبی از لایه‌های بلوک‌های مقایسه و جابجایی موازی (CAS) است که ورودی‌هایبیتی نامرتب را به خروجی‌های N-بیتی مرتب‌شده تبدیل می‌کند. هر بلوک CAS دارای دو ورودی و دو خروجی است. اگر ورودی‌ها مرتب باشند، رکوردها به صورت مستقیم به خروجی می‌روند؛ در غیر این صورت، بلوک CAS رکوردها را جابجا می‌کند. بلوک‌های CAS توسط یک مقایسه‌گر M/2 بیتی و دو مالتی‌پلکسر ۲:۱ پیاده‌سازی می‌شوند. هر رکورد به دو بخش تقسیم می‌شود: کلید و داده. کلید برای مقایسه و ترتیب‌دهی

استفاده می‌شود در حالی که داده بدون تغییر از بلوک‌های CAS عبور می‌کند. دو نوع بلوک CAS وجود دارد: افزایش‌دهنده و کاهش‌دهنده. شکل ۱(a) یک بلوک CAS افزایش‌دهنده را نشان می‌دهد. پس از مقایسه دو رکورد ورودی، رکورد کوچکتر از مالتی‌پلکسر راست و رکورد بزرگتر از مالتی‌پلکسر چپ عبور می‌کند. با استفاده از بلوک‌های CAS افزایش‌دهنده، یک شبکه مرتب‌سازی افزایشی با کوچکترین رکوردها در بالا به دست می‌آید. شکل ۱(b) یک بلوک CAS کاهش‌دهنده را نشان می‌دهد. این بلوک، رکورد کوچکتر را از مالتی‌پلکسر چپ و رکورد بزرگتر را از مالتی‌پلکسر راست عبور می‌دهد. این بلوک، شبکه مرتب‌سازی کاهشی را با بزرگترین رکوردها در بالا به دست می‌دهد. شکل ۱(c) یک بلوک CAS دوحالته را نشان می‌دهد.



شکل ۱: پیاده‌سازی در سطح بالا (قسمت بالا) و نماد شماتیک (قسمت پایین)

برای ساخت بلوک‌های CAS برای شبکه‌های مرتب‌سازی. (a) بلوک CAS افزایشی. (b) بلوک CAS کاهشی. (c) بلوک CAS دو حالته.



شکل ۲: شبکه CAS برای واحد مرتب‌سازی بیتونیک با ۸ ورودی و دارای ۳ مرحلهCAS ، ۶ گامCAS ، و ۲۴ بلوک CAS افزایشی

همانطور که از نامش پیداست، این بلوک برای کار در هر دو حالت صعودی و نزولی طراحی شده است. اگر مقدار سیگنال "Direction" صفر باشد، بلوک دوحالته مانند یک بلوک CAS افزایشی عمل می‌کند؛ در غیر این صورت، به یک بلوک CAS کاهش‌دهنده تبدیل می‌شود. تاخیر یک بلوک CAS به مقایسه‌گر و مالتی‌پلکسر ۲:۱ که به صورت سری به هم متصل شده‌اند، بستگی دارد. باید تاخیر یک گیت XOR را به این مقدار برای یک بلوک CAS دوحالته اضافه کنیم.

شبکه مرتب‌سازی بیتونیک

همانطور که قبلاً ذکر شد، یک سری از بلوک‌های CAS موازی یک شبکه مرتب‌سازی را تشکیل می‌دهند. هر بار، یک مجموعه N رکورد نامرتب وارد شبکه می‌شود. بلوک‌های CAS در هر مرحله ورودی‌های خود را مرتب می‌کنند و خروجی‌های خود را به عنوان ورودی به مرحله بعدی ارسال می‌کنند. شبکه مرتب‌سازی بیتونیک به طور گسترده‌ای برای پیاده‌سازی‌های سخت‌افزاری استفاده شده است. در یک شبکه بیتونیک با N ورودی، ما log2(N) مرحله CAS داریم. هر مرحله K رکوردی دارای log2(K) مرحله CAS با N/2 بلوک CAS موازی در هر مرحله است. شکل ۲ یک شبکه مرتب‌سازی بیتونیک با هشت ورودی را نشان می‌دهد. در مرحله اول، ما چهار بلوک CAS موازی با دو رکورد به عنوان ورودی داریم، بنابراین فقط به یک مرحله برای مرتب‌سازی آنها نیاز داریم. مرحله ۲ شامل دو مرحله متوالی است. هر مرحله رکوردهای خود را از مرحله قبلی دریافت می‌کند و آنها را مطابق با اتصالات از پیش تعریف شده بلوک‌های CAS مربوطه مرتب می‌کند. مرحله ۳ شامل سه مرحله است و مرحله آخر آن، دنباله نهایی مرتب‌شده از رکوردهای داده‌شده را خروجی می‌دهد. یک شبکه بیتونیک با N ورودی در مجموع $\log_2(N)(\log_2(N)+1)/2$ مرحله دارد. تعداد کل بلوک‌های CAS مورد نیاز برای شبکه از معادله زیر محاسبه می‌شود:

$$NumCAS = N/4 \times \log_2(N)(\log_2(N)+1) \qquad (۱)$$

برای مثال، واحد مرتب‌سازی بیتونیک با هشت ورودی نشان داده‌شده در شکل ۲ شامل $\log_2(8) = 3$ مرحله، $1/2 \times log_2(8)\left(log_2(8)+1\right) = 6$ گام و $8/4 \times log_2(8)\left(log_2(8)+1\right) = 24$ بلوک CAS افزایش‌دهنده است. منابع مورد نیاز برای پیاده‌سازی یک واحد مرتب‌سازی بیتونیک به تعداد بلوک‌های CAS بستگی دارد. همچنین، تاخیر شبکه به تعداد مراحل بستگی دارد. علاوه بر این، طبق معادله (۱)، این تاخیر تعداد ورودی‌های قابل قبول را محدود می‌کند.

۳- کارهای مرتبط

چندین راه‌حل PHSA برای بهبود فرآیند مرتب‌سازی بر اساس شبکه‌های مرتب‌سازی بیتونیک و ادغام پیشنهاد شده است. در این بخش، به معرفی برخی از این روش‌ها که بر روی FPGAها پیاده‌سازی شده‌اند، پراخته می‌شود.

Srivastava et al.: این محققان یک طراحی ترکیبی برای مرتب‌سازی در مقیاس بزرگ بر روی FPGA پیشنهاد کرده‌اند. شبکه مرتب‌سازی ادغامی از تاخیر کم و بهره‌وری حافظه بهینه بهره می‌برد، اما به دلیل کمبود موازی‌سازی در مرحله نهایی شبکه، از توان عملیاتی پایین رنج می‌برد. در مقابل، شبکه مرتب‌سازی بیتونیک توان عملیاتی بالایی دارد، اما هزینه آن تاخیر زیاد و نیازهای حافظه بالاست. نویسندگان با ترکیب شبکه‌های مرتب‌سازی ادغامی و بیتونیک، سعی در رفع نقاط ضعف ذاتی هر دو روش دارند. در این روش، مراحل اولیه از ساختار مرتب‌سازی ادغامی استفاده می‌کند، در حالی که مراحل نهایی از روش بیتونیک برای افزایش موازی‌سازی بهره می‌برد. این ترکیب، توان عملیاتی را بهبود بخشیده و مصرف حافظه را کاهش می‌دهد، البته با هزینه افزایش نیازهای منابع. همچنین، برای پیدا کردن کمترین و بیشترین ورودی‌ها، نیاز به مرتب‌سازی کامل تمامی رکوردها دارد.

Ricco et al.: این محققان کاربرد جدیدی از شبکه‌های مرتب‌سازی در مبدل‌های چندسطحی مدولار (MMC) با معرفی یک معماری جدید مبتنی بر بیتونیک برای بهبود کنترل تعادل ولتاژ خازن‌ها (CVB) ارائه داده‌اند. آنها یک روش تجزیه برای اشتراک واحدهای CAS بین زیرمدول‌های تجزیه شده معرفی کرده‌اند. این روش مصرف منابع را کاهش می‌دهد، اما افزایش سطح تجزیه باعث بدتر شدن تاخیر و توان عملیاتی شبکه مرتب‌سازی می‌شود.

Rocket Queue Architecture: این معماری ساختار صف Min/Max را بر اساس رجیسترهای شیفتی و مرتب‌سازی هپ پیاده‌سازی کرده است. رجیسترهای شیفتی به بخش‌هایی تقسیم می‌شوند که هر بخش تنها یک بلوک CAS دارد، که منجر به کاهش سطح و مصرف توان می‌شود. رکوردهای ورودی با اولین سلول هر صف مقایسه می‌شوند. اگر کوچک‌تر باشد، جابجا می‌شود، وگرنه به بخش‌های پایین‌تر حرکت می‌کند و این فرآیند تا پیدا کردن محل مناسب برای رکورد ورودی ادامه می‌یابد. مسیر بحرانی پیدا کردن کوچک‌ترین رکورد ثابت و قابل پیش‌بینی است. از سوی دیگر، اگر یک رکورد جدید وارد صف شود و از اکثر رکوردهای صف بزرگ‌تر باشد، تاخیر زیادی در فرآیند مرتب‌سازی برای جای‌گذاری رکورد جدید ایجاد می‌کند که نقطه ضعف اصلی این معماری است.

معماری درخت ادغام موازی (PMT): Song و همکاران درخت ادغام موازی (PMT) را معرفی کرده‌اند، یک شبکه مرتب‌سازی ادغام با کارایی بالا با واحدهای ادغام چندنرخی (MM) در هر مرحله. یک واحد MM شامل یک شبکه بایتونیک جزئی ورودی P و P صف FIFO است. شبکه بایتونیک جزئی P/2 از بزرگترین رکوردها را انتخاب کرده و به خروجی‌های خود می‌فرستد. معماری آن‌ها توان عملیاتی بالایی برای تعداد کمی از رکوردها دارد. اما با افزایش ورودی‌ها، مسیر بحرانی طولانی‌تر می‌شود و کارایی منابع و حافظه کاهش می‌یابد.

معماری مرتب‌سازی کم‌مصرف

Lin و همکاران یک معماری مرتب‌سازی با کارایی بالا و کم‌مصرف ارائه کرده‌اند. راه‌حل آن‌ها از یک ماژول مرتب‌سازی کم‌مصرف بهره می‌برد که با یک عملیات قطع تطبیقی دنبال می‌شود تا توان عملیاتی افزایش یابد. طول مسیر بحرانی با افزایش رکوردهای ورودی به آرامی افزایش می‌یابد که با محدودیت‌های بلادرنگ سازگار است، اما منابع مورد نیاز برای مرتب‌ساز به طور چشمگیری افزایش می‌یابد.

مرتب‌ساز سخت‌افزاری ادغامی (SHMS): SHMS یک مرتب‌ساز سخت‌افزاری ادغامی است که قطعات مرتب‌شده رکوردها را به یک توالی عمومی ترتیب می‌دهد. همانطور که قبلاً اشاره شد، مرتب‌سازی ادغامی به دلیل افزایش مسیر بحرانی روی فرکانس تأثیر منفی می‌گذارد. SHMS تلاش می‌کند این مشکل را به روش نوآورانه‌ای با محدود کردن تعداد دروازه‌های مدار متوالی به یک عدد ثابت حل کند که منجر به فرکانس ثابت می‌شود. این معماری توانسته است توان عملیاتی را نسبت به دیگر راه‌حل‌های PHSA بهبود بخشد.

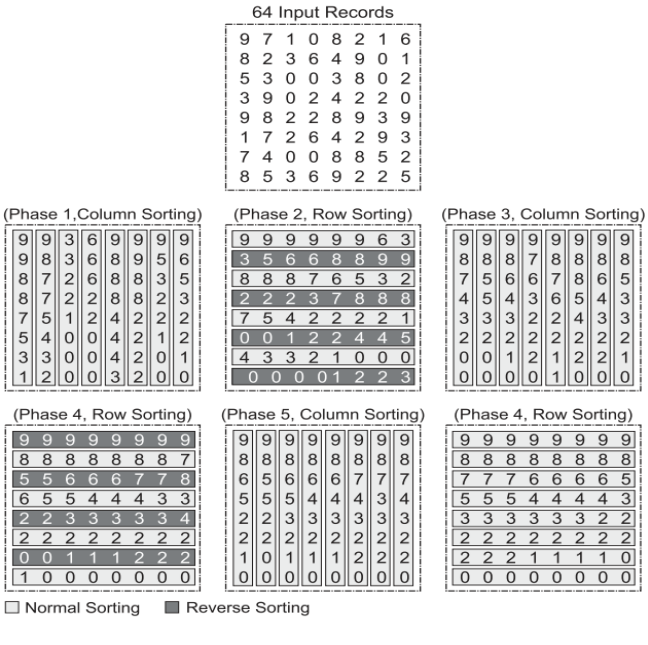
شبکه مرتب‌سازی قابل استفاده مجدد (RSN): سرعت پایین و نیازهای بالای منابع شبکه‌های مرتب‌سازی بایتونیک و حتی‌فرد، در غیاب پایپ‌لاین، Sklyarov و Skliarova را به توسعه یک روش مرتب‌سازی وادار کرد که رکوردها را به صورت تکراری مرتب می‌کند. مرتب‌سازی تکراری شامل زیرمرتب‌سازهایی است که از دو لایه واحدهای CAS متوالی تشکیل شده‌اند. زیرمرتب‌ساز از طریق رجیسترها تغذیه شده و در نهایت خروجی‌های آن به همان رجیسترها بازگردانده می‌شوند. این روش را شبکه مرتب‌سازی قابل استفاده مجدد (RSN) می‌نامیم. نویسندگان تأکید می‌کنند که با استفاده از مرتب‌سازی نرم‌افزاری و مرتب‌سازی جزئی رکوردهای ورودی، می‌توان تعداد تکرارها را کاهش داد. این روش مجموعه‌های جزئی از رکوردها را به صورت مستقل مرتب می‌کند و سپس همه مجموعه‌ها را به یک توالی واحد از رکوردهای مرتب‌شده ادغام می‌کند، با استفاده از یک روش ادغام که در نرم‌افزار پیاده‌سازی شده

است. البته با افزایش تعداد ورودی‌ها، تعداد تکرارها نیز افزایش می‌یابد و توان عملیاتی شروع به کاهش می‌کند.

پردازش یوناری: در پردازش یوناری برای طراحی یک مرتب‌ساز کم‌هزینه و مقاوم در برابر خطا استفاده شده است. پردازش یوناری یک زیرشاخه از محاسبات استوکستیک است. همه بیت‌ها دارای مقادیر معادل هستند. اعداد به صورت یک جریان بیت نشان داده می‌شوند، که اجازه می‌دهد مدارهای محاسباتی ساده و کارآمد از نظر توان و مصرف مساحت باشند. بلوک‌های CAS تنها با استفاده از یک دروازه AND و یک دروازه OR پیاده‌سازی می‌شوند. بنابراین، تاخیر و منابع مورد نیاز برای هر بلوک CAS به طور چشمگیری کاهش می‌یابد. با این حال، روش آن‌ها از توان عملیاتی پایین و تأخیر زیاد رنج می‌برد. این تأخیر به افزایش اندکی در عرض داده به خوبی پاسخ نمی‌دهد.

۴- الگوریتم مرتب‌سازی چندبعدی

همانطور که قبلاً بحث شد، محدودیت‌هایی مانند مسیر بحرانی، منابع موجود و مصرف توان بر تعداد رکوردهای ورودی قابل پردازش تاثیر می‌گذارند. برای مرتب‌سازی یک توالی بزرگ از رکوردها، ابتدا باید این توالی به بخش‌های کوچکتری تقسیم شود و هر بخش به صورت جداگانه مرتب گردد. در نهایت، برای ترکیب رکوردهای مرتب‌شده، از یک مرتب‌ساز ثانویه که شامل مجموعه‌ای از صف‌های FIFO و واحدهای مقایسه‌کننده است، استفاده می‌شود. این مرتب‌ساز ثانویه باعث افزایش تأخیر سیستم شده و منابع زیادی از FPGA را مصرف می‌کند. هدف ما این است که روشی را پیدا کنیم که بتواند بدون نیاز به مرتب‌ساز ثانویه، رکوردهای ورودی بیشتری را مدیریت کند.



شکل ۳: الگوریتم پیشنهادی دو بعدی برای مرتب‌سازی رکوردهای ورودی ماتریس ۸ × ۸

روش مرتب‌سازی

تصور کنید ما با N رکورد ورودی کار می‌کنیم، هرکدام دارای عرض M بیت هستند و این رکوردها یک ماتریس P × P را تشکیل می‌دهند که در آن P برابر با $\sqrt{N}$ است. الگوریتم مرتب‌سازی چندبعدی (MDSA) از شبکه‌های مرتب‌سازی P استفاده می‌کند تا همه رکوردها را به‌طور کامل مرتب کند. به عنوان مثال، در شکل ۳ نمایش داده شده است که الگوریتم MDSA چگونه ۶۴ رکورد ورودی که یک ماتریس ۸ × ۸ را تشکیل می‌دهند، را به‌صورت مرتب می‌کند.

مراحل اجرای الگوریتم

شبکه مرتب‌سازی می‌تواند در دو حالت عمل کند:

۱. حالت مرتب‌سازی عادی: در این حالت، شبکه مرتب‌سازی رکوردها را به ترتیب نزولی مرتب می‌کند.

۲. حالت مرتب‌سازی معکوس: در این حالت، شبکه مرتب‌سازی رکوردها را به ترتیب صعودی مرتب می‌کند.

الگوریتم MDSA شامل شش مرحله است:

۱. مرتب‌سازی ستون‌ها: شبکه‌های فرد و زوج در حالت مرتب‌سازی عادی عمل می‌کنند.

۲. مرتب‌سازی سطرها: شبکه‌های زوج در حالت مرتب‌سازی معکوس و شبکه‌های فرد در حالت مرتب‌سازی عادی عمل می‌کنند.

۳. مرتب‌سازی ستون‌ها: شبکه‌های فرد و زوج در حالت مرتب‌سازی عادی عمل می‌کنند.

۴. مرتب‌سازی سطرها: شبکه‌های فرد در حالت مرتب‌سازی معکوس و شبکه‌های زوج در حالت مرتب‌سازی عادی عمل می‌کنند.

۵. مرتب‌سازی ستون‌ها: شبکه‌های فرد و زوج در حالت مرتب‌سازی عادی عمل می‌کنند.

۶. مرتب‌سازی سطرها: شبکه‌های فرد و زوج در حالت مرتب‌سازی عادی عمل می‌کنند.

در هر مرحله، هر شبکه مرتب‌سازی به‌طور مستقل هشت رکورد تخصیص داده شده خود را مرتب می‌کند. در پایان مرحله ششم، تمام رکوردها مرتب شده و به‌صورت یک آرایه یک‌بعدی به خروجی ارسال می‌شوند.

در هر مرحله، هر شبکه مرتب‌سازی هشت رکورد تخصیص داده شده به خودش را مستقل از دیگر شبکه‌ها مرتب می‌کند. در مرحله اول، رکوردهای ستون‌ها به شبکه‌های مرتب‌سازی اختصاص داده می‌شوند. شبکه‌های مرتب‌سازی در حالت مرتب‌سازی عادی عمل می‌کنند و رکوردهای خود را به ترتیب نزولی برای هر ستون مرتب می‌کنند.

در مرحله دوم، رکوردهای ردیف‌ها به شبکه‌های مرتب‌سازی تخصیص داده می‌شوند. شبکه‌های فرد و زوج به ترتیب در حالت‌های مرتب‌سازی عادی و معکوس قرار دارند. جابجایی رکوردها در ردیف‌های مجاور، با استفاده از دو حالت مرتب‌سازی عادی و معکوس، امکان مقایسه تمام رکوردها را در مراحل بعدی فراهم می‌کند.

در مرحله سوم، شبکه‌های مرتب‌سازی رکوردها را به عنوان ستون‌ها دریافت و مرتب می‌کنند در حالت مرتب‌سازی عادی. پس از مرحله سوم، مرتب‌سازی جزئی انجام می‌شود و رکوردهای بزرگتر و کوچکتر به ترتیب در عناصر ماتریس بالایی و پایینی قرار می‌گیرند.

مرحله چهارم تکرار مرحله دوم است و مرحله پنجم مشابه مراحل اول و سوم است. در مرحله ششم، رکوردها به عنوان ردیف‌ها به شبکه‌های مرتب‌سازی اختصاص داده می‌شوند و یک بار دیگر به صورت عادی مرتب می‌شوند. در پایان مرحله ششم، تمام رکوردها مرتب شده و به عنوان یک آرایه یک بعدی به خروجی ارسال می‌شوند.

ما تمایل داریم از MDSA برای مرتب‌سازی یک ماتریس n بعدی استفاده کنیم. فرض کنید a(i,j,...,n) یک عنصر ماتریس است و i × n ×... × j اندازه‌های متناظر آن‌ها را نشان می‌دهد.

قضیه ۱:

رکوردهای یک ماتریس n بعدی می‌توانند به یک آرایه یک بعدی مرتب شده تبدیل شوند، اگر روابط زیر برقرار باشند (۱ ≤ n ≤ w).

در بعد w ام، رکورد a_m باید بزرگتر از رکورد $a_{(m+1)}$ باشد $1 \le m \le P$ و $P = ([n]N)^{1/2}$ در بعد w ام، رکورد $a_{(c,P)}$ از این بعد باید بزرگتر از رکورد $a_{(c+1,1)}$ باشد، برای $1 < c \le P$ در $a_{(k,l)}$، k بعد w−1 ام و l بعد w ام است.

اثبات:

اگر بخواهیم رکوردهای ماتریس $A_{(i, j,...,w)}$ را در آرایه یک بعدی B قرار دهیم، از معادله زیر استفاده می‌کنیم:

$$B[P^{n-1}w+...+Pj+i] = A(i, j,..., w) \qquad (۲)$$

در این معادله، i, j, ... , w می‌توانند در محدوده بین ۱ و P تغییر کنند. معادله زیر برای بررسی ترتیب نزولی رکوردها در آرایه B استفاده می‌شود:

$$B[m] \ge B[m+1], 0 < m < P \qquad (۳)$$

طبق این معادله، در هر بعد، رکوردها باید به ترتیب نزولی مرتب شوند و آخرین رکورد هر بعد باید بزرگتر از اولین رکورد بعدی باشد.

با توجه به مجموعه $\{\bar{x}, \underline{x}, x\}$ به عنوان نمایش حالت‌های مرتب‌سازی در بعد x، در حالت x، تمام شبکه‌های مرتب‌سازی در حالت عادی عمل می‌کنند. در حالت $\bar{x}$، شبکه‌های مرتب‌سازی فرد/زوج به ترتیب در حالت‌های عادی/معکوس عمل می‌کنند، و در حالت x، شبکه‌های مرتب‌سازی فرد/زوج به ترتیب در حالت‌های معکوس/عادی عمل می‌کنند. برای مثال، برای مرتب‌سازی یک ماتریس دو بعدی (i نشان‌دهنده ردیف و j نشان‌دهنده ستون است)، به شش مرحله نیاز داریم که به صورت $\{j\ \bar{i},\ j\ \underline{i},\ j\ i\}$ نمایش داده می‌شوند.

ما یک ماتریس دوبعدی با ابعاد P × P را در نظر می‌گیریم که در آن a(i, j) < a(i+1, j) و a(i, j) < a(i, j+1) است. در مرحله اول (با شاخص j)، ستون‌های ماتریس به ترتیب نزولی مرتب می‌شوند و نصف بالایی هر ستون شامل P/2 بزرگترین رکوردها و نصف پایینی شامل P/2 کوچکترین رکوردها می‌شود. در مرحله دوم (با شاخص $\bar{i}$)، سطرهای فرد و زوج ماتریس به ترتیب نزولی و صعودی مرتب می‌شوند. پس از این مرحله، رکورد a(P,P) به سطر و ستون اول منتقل می‌شود، اما رکورد a(P,P−1) در سطر دوم و ستون چهارم باقی می‌ماند. این جابجایی‌ها باعث می‌شوند که بزرگترین رکوردها در ماتریس پراکنده شوند، به طوری که امکان مقایسه تمام رکوردها در مراحل بعدی فراهم می‌شود. در مرحله سوم (با شاخص j)، ستون‌ها دوباره به ترتیب نزولی مرتب می‌شوند و بزرگترین رکوردها در بالا و کوچکترین رکوردها در پایین ماتریس قرار می‌گیرند. در این مرحله، تمام رکوردها در سطرها و ستون‌های خود مرتب شده‌اند، اما هدف ما این است که رکوردهای مرتب‌شده را به صورت یک آرایه یک‌بعدی مرتب‌شده خروجی بگیریم. برای تحقق شرایط قضیه (۱)، معادله زیر برای مرتب‌سازی دوبعدی به‌دست می‌آید (0 < i ≤ P):

$$\begin{cases} a_{i,j} \ge a_{i,j+1}, 0 < j < P \\ a_{i,P} \ge a_{i+1,1}, j = P \end{cases} \qquad (۴)$$

مراحل بعدی برای تحقق این شرایط در نظر گرفته شده‌اند. در مرحله چهارم (با شاخص i)، سطرهای فرد و زوج ماتریس به ترتیب صعودی و نزولی مرتب می‌شوند. دو مرحله بعدی (با شاخص‌های ji) سطرها و ستون‌های ماتریس را به ترتیب نزولی مرتب می‌کنند. در نهایت، شرط (۴) برقرار شده و فرآیند مرتب‌سازی تکمیل می‌شود.

ما می‌توانیم مراحل مرتب‌سازی را برای یک ماتریس n-بعدی با معادله زیر خلاصه کنیم که خاصیت توزیعی دارد اما خاصیت جابجایی ندارد.

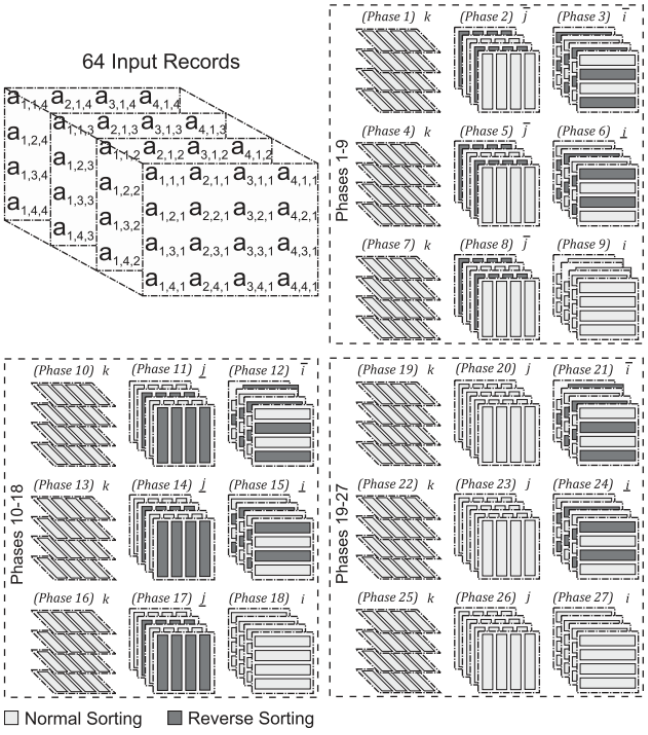
$$\{n\{\bar{q}, \underline{q}, q\}...\{\bar{k}, \underline{k}, k\}\}\{\bar{j}, \underline{j}, j\}\{\bar{i}, \underline{i}, i\}\} \qquad (۵)$$

در این معادله، حالت‌های مرتب‌سازی در هر بعد تغییر می‌کنند به جز بعد آخر که حالت مرتب‌سازی باید ثابت باشد. برای

مرتب‌سازی یک ماتریس n-بعدی، به $n\times3^{(n-1)}$ مراحل مرتب‌سازی نیاز داریم. به عنوان مثال، برای ماتریس‌های دوبعدی و سه‌بعدی به ترتیب ۶ و ۲۷ مرحله لازم است. افزایش مراحل باعث افزایش تأخیر می‌شود، اما تعداد بلوک‌های CAS مورد نیاز برای پیاده‌سازی مراحل بسیار کمتر از روش‌های معمولی است. اگر تعداد رکوردهای ورودی در مرتب‌سازی اصلی ثابت بماند، واحدهای شبکه مرتب‌سازی با افزایش ابعاد ماتریس ساده‌تر می‌شوند.

شکل ۴ مراحل مرتب‌سازی MDSA برای یک ماتریس سه‌بعدی A4,4,4 را نشان می‌دهد که برای مرتب‌سازی ۶۴ رکورد ورودی در نظر گرفته شده است. مطابق معادله (۵)، مرتب‌سازی یک ماتریس سه‌بعدی ۲۷ مرحله طول می‌کشد. در بعد اول (با شاخص i)، حالت مرتب‌سازی برای هر سطر تغییر می‌کند. در بعد دوم (با شاخص j)، حالت مرتب‌سازی برای هر صفحه تغییر می‌کند و ستون‌های هر صفحه دارای همان حالت مرتب‌سازی هستند. در بعد سوم (با شاخص k)، حالت مرتب‌سازی برای تمام مراحل ثابت است. فرآیند مرتب‌سازی ماتریس سه‌بعدی می‌تواند با فرآیند مرتب‌سازی دوبعدی نشان داده شده در شکل ۳ جایگزین شود. انتخاب سه‌بعدی نسبت به دوبعدی منابع سخت‌افزاری را کاهش می‌دهد اما ۴,۵ برابر مراحل مرتب‌سازی بیشتری نیاز دارد.

این الگوریتم نیاز به سخت‌افزار مرتب‌سازی ثانویه را از بین می‌برد و تعداد رکوردهای ورودی بر تعداد مراحل تأثیری ندارد. اندازه ماتریس ورودی و تعداد شبکه‌های مرتب‌سازی و مراحل مربوطه افزایش می‌یابد، اما مراحل الگوریتم ثابت می‌ماند (شش مرحله در ماتریس دوبعدی). ادغام مراحل پایپ‌لاین در شبکه‌های مرتب‌سازی باعث می‌شود زمان اجرای ثابتی داشته باشیم، به‌طوری‌که یک زمان‌بند واقعی می‌تواند به راحتی زمان اجرای وظایف مرتب‌سازی را محاسبه و برنامه‌ریزی کند.



شکل ۴: الگوریتم پیشنهادی سه‌بعدی برای مرتب‌سازی رکوردهای ورودی ماتریس ۴ × ۴ × ۴

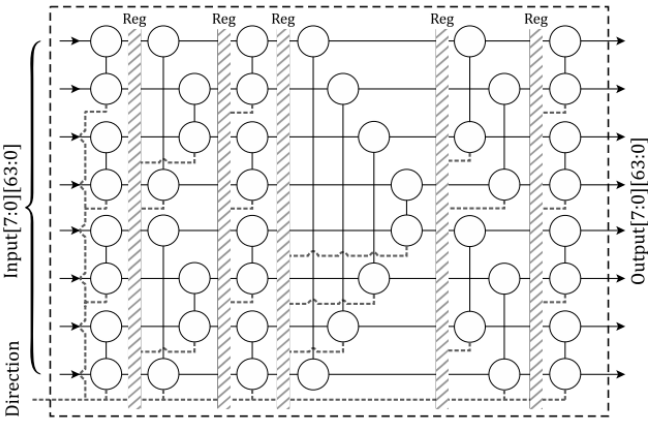
این الگوریتم برای سایر ماتریس‌های غیردوگانه نیز قابل اعمال است، اما ماتریس مربع شکل بهینه‌ترین فرم برای پیاده‌سازی سخت‌افزاری است. اگر تعداد رکوردهای ورودی بین $2^{2k}+1$ و $2^{2(k+1)}$ برای $k = 1$، ۲، ۳ و ... باشد، از یک مرتب‌ساز که قادر به مرتب‌سازی $2^{2(k+1)}$ رکورد است استفاده می‌کنیم. به عنوان مثال، تأخیر در مرتب‌سازی تعداد رکوردها بین ۱۷ و ۶۴ یکسان است، اما اگر تعداد رکوردهای ورودی از این حد بیشتر شود، تأخیر کلی به دلیل افزایش تعداد مراحل شبکه‌های مرتب‌سازی افزایش می‌یابد.

در مرتب‌سازی داده‌های بزرگ، فرض می‌کنیم که یک ابزار نرم‌افزاری رکوردهای ورودی را به بخش‌هایی تقسیم می‌کند و سپس هر بخش را با استفاده از MDSA مرتب می‌کند. ابزار نرم‌افزاری رکوردها را بر اساس الگوریتم‌های مرتب‌سازی مانند بیتونیک به بخش‌ها تخصیص می‌دهد و این فرآیند تا زمانی که تمام رکوردها مرتب شوند ادامه می‌یابد.

کاربرد دیگر این الگوریتم به دست آوردن رکوردهای Min/Max در دو مرحله است. طبق شکل ۳، در مرحله اول بزرگترین رکوردها در اولین سطر و کوچکترین رکوردها در آخرین سطر قرار می‌گیرند. در مرحله دوم، بزرگترین رکورد در اولین سطر و ستون اول و کوچکترین رکورد در آخرین سطر و ستون اول قرار می‌گیرد.

۵- روش پیشنهادی

در این بخش، پیاده‌سازی سخت‌افزاری MDSA برای یک ماتریس ورودی دو بعدی را توضیح می‌دهیم. برای تشریح طراحی ما، ابتدا به بررسی طراحی شبکه بیتونیک پایپ‌لاین دو حالته (DPBN) می‌پردازیم.



شکل ۵: DPBN با ۸ ورودی

شبکه بیتونیک پایپلاین دو حالته

افزایش تعداد رکوردهای ورودی در یک شبکه مرتب‌سازی بایتونیک باعث افزایش تعداد مراحل شبکه می‌شود. این امر مسیر بحرانی(critical path) را طولانی‌تر کرده و تأخیر شبکه (latency) را افزایش می‌دهد. بنابراین، از تکنیک پایپلاین (pipeline) برای کاهش مسیر بحرانی استفاده می‌شود.

تعداد مراحل پایپلاین در یک شبکه بایتونیک دو حالته (DPBN) برابر با تعداد مراحل آن است که برابر است با :

$$1/2\log_2(N)(\log_2(N)+1)$$

شکل ۵ یک شبکه بایتونیک دو حالته با هشت ورودی را نشان می‌دهد. این شبکه می‌تواند تا هشت رکورد ورودی با عرض ۶۴ بیت را در هر سیکل ساعت دریافت کرده و رکوردهای مرتب‌شده را پس از شش مرحله پایپلاین به خروجی تحویل دهد.

قسمت کنترل، سیگنال "direction" را ارسال می‌کند تا حالت بین مرتب‌سازی عادی و معکوس تنظیم شود.

در این روش، با استفاده از پایپلاین، می‌توان تأثیر افزایش رکوردهای ورودی بر تأخیر کلی شبکه را به حداقل رساند.

سخت‌افزار اصلی مرتب‌سازی پیشنهادی

مطابق با الگوریتم پیشنهادی برای ماتریس ورودی دوبعدی (D-۲)، به تعداد P مرتب ساز ستون وP مرتب‌ساز سطر نیاز داریم.

مرتب‌سازهای سطر باید شامل هر دو نوع بلوک CAS کاهشی و افزایشی باشند. ما شبکه مرتب‌سازی را با استفاده از P واحد DPBN طراحی می‌کنیم تا نیازهای سخت‌افزاری را کاهش دهیم.

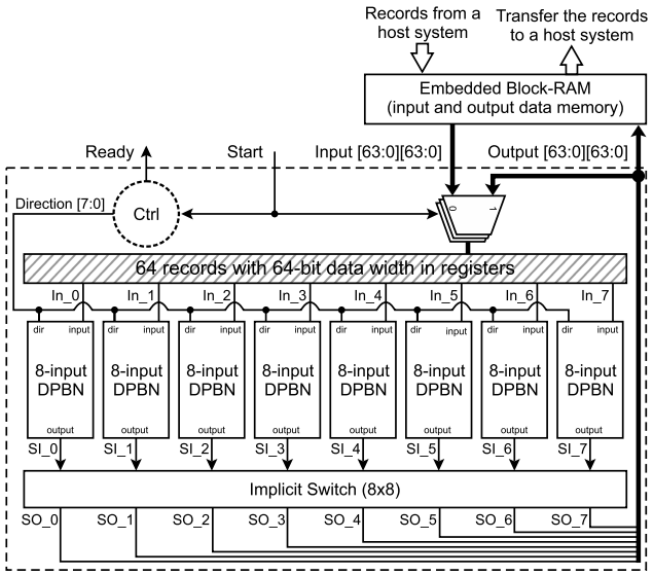
شکل ۶ معماری مرتب‌سازی پیشنهادی به نام RTHS (real-time hardware sorter) برای مرتب‌سازی رکوردهای ماتریس ۸×۸ را نشان می‌دهد. این معماری از هشت DPBN، یک سوئیچ ضمنی ۸×۸ (implicit switch)، رجیسترها و یک کنترل کننده تشکیل شده است.

مدار مالتی‌پلکسینگ توسط یک سوئیچ ضمنی (implicit switch) پیاده‌سازی شده است. این سوئیچ موقعیت سطرها و ستون‌ها در یک ماتریس را جابه‌جا می‌کند.

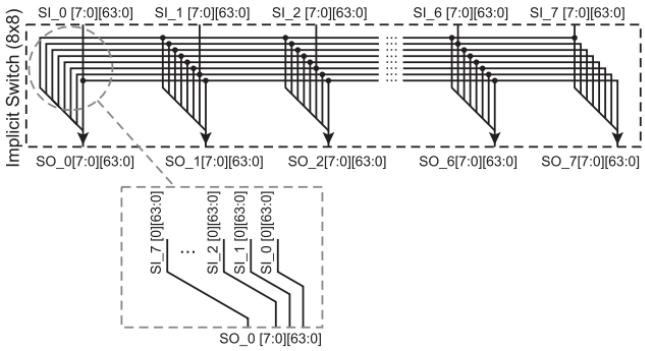
این سوئیچ به‌طور ضمنی در داخل شبکه مرتب‌سازی پیاده‌سازی شده و نیاز به سخت‌افزار اضافی ندارد.

در شکل ۷، هر خط افقی به یک خط ورودی متصل است که یک مجموعه از هشت رکورد ۶۴ بیتی را منتقل می‌کند. سپس هر خط افقی، رکوردهای خود را به هشت خط خروجی عمودی توزیع می‌کند. به این ترتیب، اولین رکورد از هر ورودی با هم ترکیب شده و به صورت یک مجموعه هشت‌تایی از طریق اولین خروجی ارسال می‌شود. این فرآیند برای تمامی خروجی‌ها به همین شکل انجام می‌شود.

در ابتدا، فرآیند با بارگذاری ۶۴ رکورد با عرض داده ۶۴ بیت در رجیسترهای اولیه از حافظه‌های دسترسی تصادفی بلوکی (BRAMs) آغاز می‌شود و سپس سیگنال "Start" فعال می‌شود. در سیکل بعدی، رکوردهای موجود در رجیسترهای اولیه به هشت واحد DPBN برای مرتب‌سازی اختصاص داده می‌شوند. واحد کنترل، سیگنال‌های "Direction" را به واحدهای DPBN ارسال می‌کند تا الگوریتم مرتب‌سازی پیشنهادی اجرا شود. هر بیت از سیگنال "Direction" مشخص می‌کند که واحد DPBN مربوطه در حالت مرتب‌سازی عادی ("۰") یا معکوس ("۱") عمل کند. سوئیچ ضمنی(implicit switch)، سطرها و ستون‌های ماتریس را جابه‌جا کرده و آن‌ها را به رجیسترهای اولیه بازمی‌گرداند. واحد کنترل، فرآیند شش مرحله‌ای مرتب‌سازی را با صدور سیگنال‌های کنترلی مناسب مدیریت می‌کند.



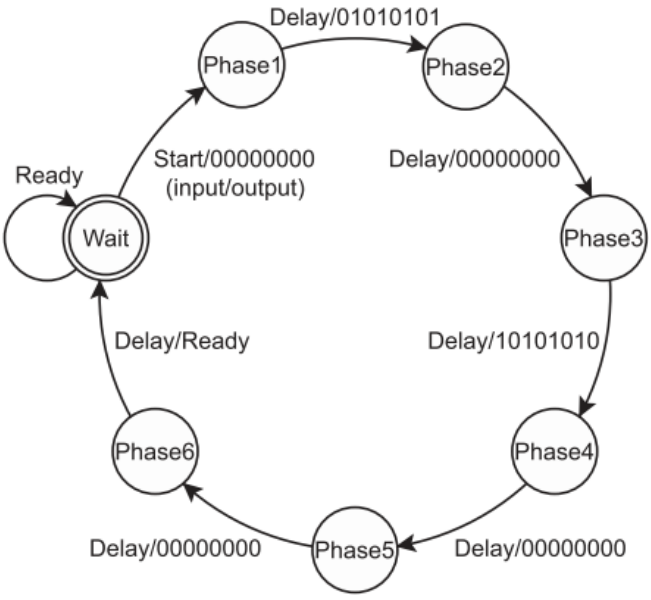
شکل ۶: طراحی RTHS برای مرتب‌سازی رکوردهای ماتریس ۸ × ۸



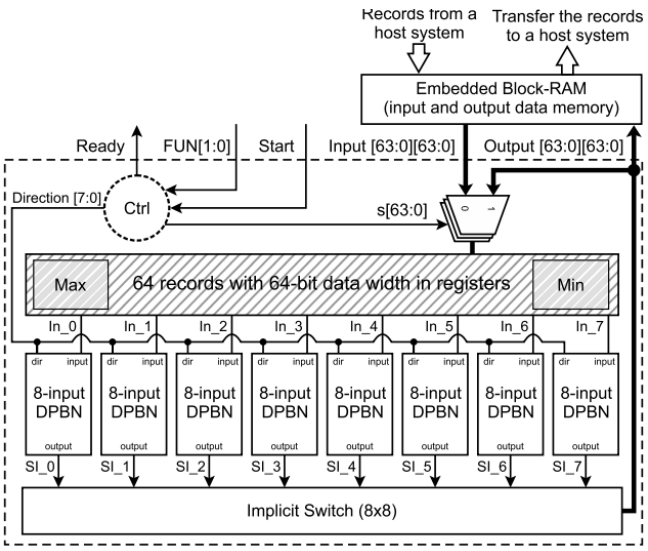
شکل ۷: طراحی سوییچ Implicit

در نهایت، واحد کنترل سیگنال "Ready" را صادر می‌کند تا نشان دهد که رکوردهای موجود در رجیسترهای اولیه مرتب شده‌اند.

شکل ۸ FSM واحد کنترل را نشان می‌دهد. مرتب‌ساز در ابتدا در حالت "Wait" است. زمانی که سیگنال "Start" فعال می‌شود، مقدار اولیه به سیگنال‌های "Direction" اختصاص داده می‌شود. پس از هر شش سیکل ساعت سیگنال "Delay" فعال می‌شود. حالات FSMو سیگنال‌های "Direction" مطابق شکل ۸ تغییر می‌کنند. در نهایت، زمانی که سیگنال "Ready" فعال می‌شود، مرتب‌ساز به حالت "Wait" باز می‌گردد.



شکل ۸: FSM واحد کنترل



شکل ۹: طراحی RTHS بهینه شده

سخت‌افزار مرتب‌سازی توسعه‌یافته پیشنهادی

در برخی موارد، نیاز است که رکوردهای حداکثر و حداقل را به سریع‌ترین شکل ممکن پیدا کنیم. همان‌طور که در شکل ۳ نشان داده شده است، تنها دو مرحله از الگوریتم پیشنهادی برای یافتن رکوردهای حداکثر و حداقل در یک ماتریس ورودی دوبعدی کافی است.

شکل ۹ یک معماری بهبود یافته برای به‌دست آوردن رکوردهای حداکثر و حداقل را نشان می‌دهد. در ابتدا، ۶۴ رکورد ورودی با عرض ۶۴ بیت به مرتب‌ساز اختصاص داده می‌شود. پس از دو مرحله، رکوردهای حداکثر و حداقل به ترتیب در رجیسترهای چپ‌ترین و راست‌ترین قرار می‌گیرند. سپس، سیگنال "Ready" فعال

شده و رکوردهای میانی با ۶۲ رکورد جدید در رجیسترهای اولیه جایگزین می‌شوند. این فرآیند ادامه خواهد داشت تا زمانی که تمام رکوردهای داده بزرگ به مرتب‌ساز تغذیه شوند. در نهایت، دو رجیستر جانبی، رکوردهای حداقل و حداکثر داده بزرگ را نگه می‌دارند.

معماری نشان‌داده‌شده در شکل ۹ می‌تواند برای صف‌هایMin/Max (Min/Max queue) نیز استفاده شود. فرض کنید که مقادیر ددلاین هر تسک در رکوردهای ورودی قرار دارد. هنگامی که یک تسک جدید به این صف می‌رسد، تسکی با ددلاین حداکثر برای جایگزینی تسک جدید حذف می‌شود. با این حال، اگر ددلاین تسک جدید بزرگتر از ددلاین حداکثر موجود در صف Min/Max باشد، به صف اضافه نمی‌شود.

صف Min/Max در دو حالت به‌روزرسانی می‌شود: ۱) تسک جدید پذیرفته می‌شود و ۲) تسکی با ددلاین حداقل به یک پردازنده برای اجرا اختصاص داده می‌شود.

مرتب‌ساز می‌تواند برای سه کاربرد مختلف با تغییر حالت‌های کنترل کننده استفاده شود: ۱) مرتب‌سازی عمومی؛ ۲) یافتن رکوردهای حداکثر و حداقل در داده‌های بزرگ؛ و ۳) ایجاد صف‌های Min/Max شکل ۹ معماری کلی RTHS را نشان می‌دهد که قابلیت تغییر کاربردهای مرتب کننده با دریافت سیگنال‌های مختلف "FUN" را دارد. حالت مرتب‌سازی عمومی با "۰۰" نشان داده می‌شود، به این معنی که تمام رکوردهای ورودی پس از شش مرحله مرتب می‌شوند. اگر مقدار FUN برابر با "۰۱" باشد، مرتب کننده رکوردهای حداکثر و حداقل را در داده‌های بزرگ پیدا می‌کند. در نهایت، اگر مقدار FUN برابر با "۱۰" باشد، مرتب کننده به حالت صف Min/Max (Min/Max queue) تغییر می‌کند، که در این حالت، تنها رکورد بزرگتر در صف تحت شرایط خاصی جایگزین می‌شود.

تحلیل مرتب‌سازی پیشنهادی

تأخیر کلی شبکه مرتب‌سازی به حداکثر تأخیر واحد CAS-Dual و مسیریابی سوئیچ ضمنی(implicit switch) در FPGA بستگی دارد. بنابراین، اگر مسیریابی سوئیچ ضمنی ایده‌آل باشد، تأخیر مرتب کننده تقریباً ثابت خواهد بود. استفاده از حافظه نیز اهمیت زیادی دارد و به سه عامل بستگی دارد: ۱) تعداد واحدهای DPBN۲)تعداد مراحل پایپلاین ۳) تعداد رجیسترهای هر مرحله پایپلاین

تعداد رجیسترهای مورد نیاز در هر مرحله پایپلاین از معادله زیر به دست می‌آید:

$$block_{reg} = M \times P \quad (۶)$$

که در آن M عرض داده و P تعداد رکوردها در یک سطر یا ستون است.

تعداد رجیسترهای مورد نیاز یک واحد DPBN:

$$bitonic_{reg} = block_{reg} \times \left(1/2 \times log_2(P)\left(log_2(P) + 1\right)\right) \quad (۷)$$

تعداد کل رجیسترهای شبکه RTHS:

$$total_{reg} = \left(bitonic_{reg} \times P\right) + M \times N \quad (۸)$$

تعداد رجیسترهای مورد نیاز برای conventional bitonic sorting network (CBSN):

$$CBSN_{reg} = M \times N \times \left(1/2 \times log_2(N)\left(log_2(N) + 1\right)\right) \quad (۹)$$

می‌توان نتیجه گرفت که RTHS نسبت به راه حل CBSN حافظه کمتری استفاده می‌کند.

جدول ۱ طراحی ما را با CBSN مقایسه می‌کند. تأخیر CBSN به عمق شبکه مرتب‌سازی بستگی دارد.

جدول ۱: مقایسه طراحی برای مرتب‌سازی عناصرN گانه

Design	Proposed	Conven. Bitonic
Latency	$Max(delay_{CASDual}, Routing)$	$\log_2 N \times delay_{CAS}$
Memory	$M \times P^2 \times \log_2^2(P)$	$M \times N \times \log_2^2(N)$
Throughput	N	N

۶- نتایج تجربی و تحلیل‌ها

تنظیمات عملی

طراحی RTHS مقاله بررسی شده بر روی یک FPGA مدل Virtex-7 (XC7VX485T, speed grade −2L) پیاده‌سازی شده است. این طراحی با استفاده از زبان توصیف سخت‌افزار Verilog نوشته شده است. این دستگاه دارای 360 BRAM دو پورتی (هرکدام ۳۶ کیلوبایت) و 303000CLB است. همچنین از نرم‌افزار Vivado برای سنتز و مسیریابی استفاده شده است.

معیارهای ارزیابی به شرح زیر هستند:

۱. استفاده از منابع(Resource Utilization): تعداد منابع مورد نیاز برای مرتب‌ساز، مانند جداول جستجو (LUTs)، رجیسترها و فرکانس عملیاتی.

۲. Throughput: تعداد بایت‌های مرتب‌شده در واحد زمان (گیگابایت در ثانیه).

۳. زمان اجرا(Execution Time): زمان مورد نیاز برای مرتب‌سازی یک توالی از رکوردهای ورودی.

۴. کارایی حافظه(Memory Efficiency): توان عملیاتی به دست آمده تقسیم بر مصرف حافظه روی تراشه (به بیت). به‌ویژه به منطقه

بالا-چپ نمودار کارایی حافظه که نشان‌دهنده بیشترین کارایی همراه با کمترین استفاده از حافظه است.

۵. مصرف توان(Power Consumption): مقدار توان مصرفی به وات در واحد زمان.

تحلیل استفاده از منابع

در این بخش، استفاده از منابع طراحی RTHS با CBSN و سایر روش‌های معمول مقایسه می‌شود. جدول II و شکل ۱۰ نتایج مقایسه RTHS با CBSN را نشان می‌دهند که در آن تعداد مراحل پایپلاین برابر با تعداد مراحل شبکه (CBSN1) و تعداد مراحل پایپلاین برابر با تعداد استیج‌های شبکه (CBSN2) برای ۱۶، ۶۴ و ۲۵۶ رکورد ورودی در نظر گرفته شده‌اند.

تعداد رکوردهای ورودی به 2^n محدود می‌شود، جایی که RTHS می‌تواند به بهینه‌ترین پیاده‌سازی سخت‌افزار دست یابد. این طراحی همچنین برای شبکه مرتب‌سازی بایتونیک با قالب ورودی 2^n مناسب است. با این حال، طراحی RTHS از اندازه‌های مختلف برای ماتریس ورودی پشتیبانی می‌کند.

تعداد بلوک‌های CAS در CBSN1 و CBSN2 از معادله (۱۰) و در RTHS از معادله زیر به دست می‌آید:

$$Num_{CAS-RTHS} = Num_{DPBN}\left(D/4 \times log_2 D\left(log_2 D + 1\right)\right) \quad (۱۰)$$

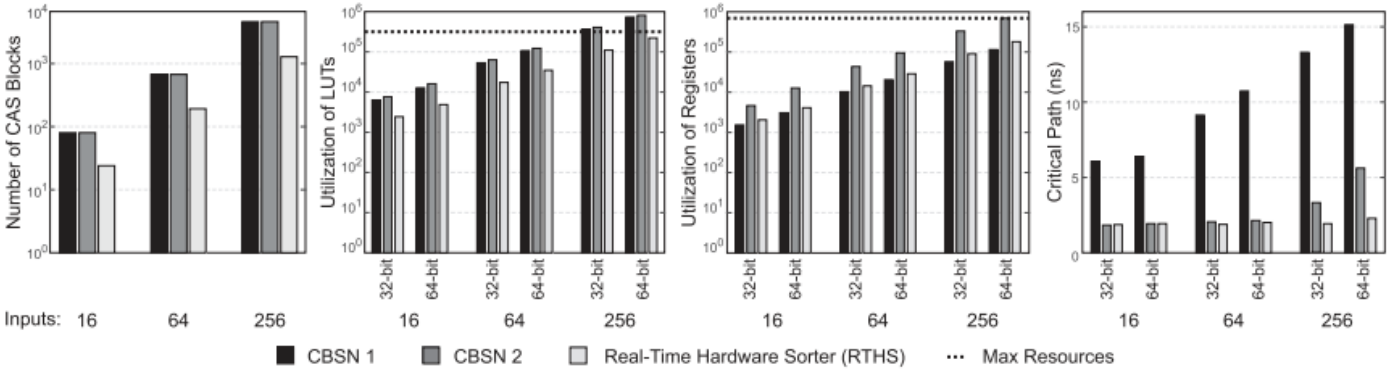
تعداد Num_{DPBN} نشان‌دهنده تعداد واحدهای DPBN و D نشان‌دهنده تعداد ورودی‌ها/خروجی‌ها در یک واحد DPBN است. مطابق جدول II، طراحی RTHS به طور متوسط ۷۱,۲٪ بلوک‌های CAS کمتری نسبت به CBSN1 نیاز دارد. بلوک‌های CAS دو حالته مورد استفاده در RTHS نسبت به بلوک‌های CAS افزایشی/کاهشی منابع بیشتری مصرف نمی‌کنند، زیرا در همان LUT پیاده‌سازی می‌شوند.

همان‌طور که در بخش قبلی توضیح داده شد، برای مقایسه منصفانه، فرض کردیم که در شبکه مرتب‌سازی بایتونیک یک مرحله پایپلاین بین هر دو مرحله DPBN قرار داده شود. همچنین CBSN2 برای نشان دادن افزایش زیاد تعداد رجیسترهای مورد نیاز در نظر گرفته شده است. در CBSN2، یک مرحله پایپلاین بین هر دو گام شبکه CBSN قرار داده شده است تا مزیت استفاده از طراحی RTHS از نظر مصرف منابع بهتر مشخص شود.

همان‌طور که می‌دانیم، حداکثر فرکانس عملیاتی به حداقل دوره زمانی ساعت‌های مدار مرتب‌سازی بستگی دارد که به طولانی‌ترین مسیر بحرانی (critical path) طراحی محدود می‌شود. نیازهای بالای منابع CBSN1 برای ۶۴ رکورد ورودی و بیشتر، به دلیل افزایش تعداد مراحل، تا حدی مزایای پایپلاینینگ را بی‌اثر می‌کند. تأثیر افزایش مراحل پایپلاین مستقیماً بر مسیر بحرانی (critical path) و دوره زمانی حداقل یک سیکل ساعت تأثیر می‌گذارد. رویکرد مقاله بررسی شده این امکان را فراهم کرده است که یک مرحله پایپلاین بین هر گام از واحدهای DPBN قرار دهیم، که منجر به کوتاه‌تر شدن مسیر بحرانی (critical path) به میزان ۸۵,۴٪ و ۶۹,۹٪ در بهترین و بدترین حالت در مقایسه با CBSN1 شده است. در پاسخ به افزایش تعداد رکوردهای ورودی، می‌توانیم به سادگی واحدهای DPBN اضافی را به طراحی خود اضافه کرد. به دلیل ماهیت موازی معماری ما، مسیر بحرانی (critical path) بدون تغییر باقی می‌ماند که منجر به زمان اجرای قابل پیش‌بینی و عملکرد بهتر در برنامه‌های زمان واقعی می‌شود.

جدول ۲: مقایسه طراحی RTHS، CBSN با تعداد مراحل خط پایپلاین برابر با تعداد مراحل شبکه (CBSN1) و CBSN با تعداد مراحل خط پایپلاین برابر با تعداد گام‌های شبکه (CBSN2)

# of I/O	# of DPBN	# of I/O in DPBN	CAS blocks			Pipeline stages			Data width	LUTs			Registers			Critical path (ns)			Frequency (MHz)		
			CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS		CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS
16	4	4	80	80	24	4	10	4	32-bit	6,344 (2%)	7,058 (2.3%)	2,437 (0.8%)	1,536 (0.25%)	5,632 (0.93%)	2,051 (0.33%)	6.061	1.822	1.871	151.2	548.7	534.4
									64-bit	12,744 (4%)	14,098 (4.7%)	4,869 (1.6%)	3,072 (0.5%)	11,264 (1.8%)	4,099 (0.6%)	6.397	1.942	1.936	156.3	514.8	516.4
64	8	8	672	672	192	6	21	6	32-bit	53,376 (17%)	59,728 (20%)	17,413 (5%)	10,240 (1%)	45,060 (7.4%)	14,340 (2%)	9.138	1.923	1.886	109.4	520	530.2
									64-bit	107,136 (35%)	113,040 (37%)	34,823 (11%)	20,480 (3%)	90,112 (15%)	28,675 (4%)	10.27	2.025	2.013	97.3	493.8	496.7
256	16	16	4,608	4,608	1,280	8	36	10	32-bit	365,696 (120%)	379,340 (125%)	110,605 (36%)	57,344 (9%)	303,104 (50%)	90,115 (14%)	13.299	3.193	1.932	75.2	313.2	517.6
									64-bit	736,128 (242%)	758,339 (250%)	221,197 (72%)	114,688 (18%)	606,208 (99%)	180,227 (29%)	15.138	5.631	2.289	66	177.5	436.8



شکل ۱۰: تعداد بلوک‌های CAS، LUTها، ثبات‌ها و مسیر بحرانی (ns) گزارش شده برای طراحی‌های CBSN1، CBSN2، و RTHS.

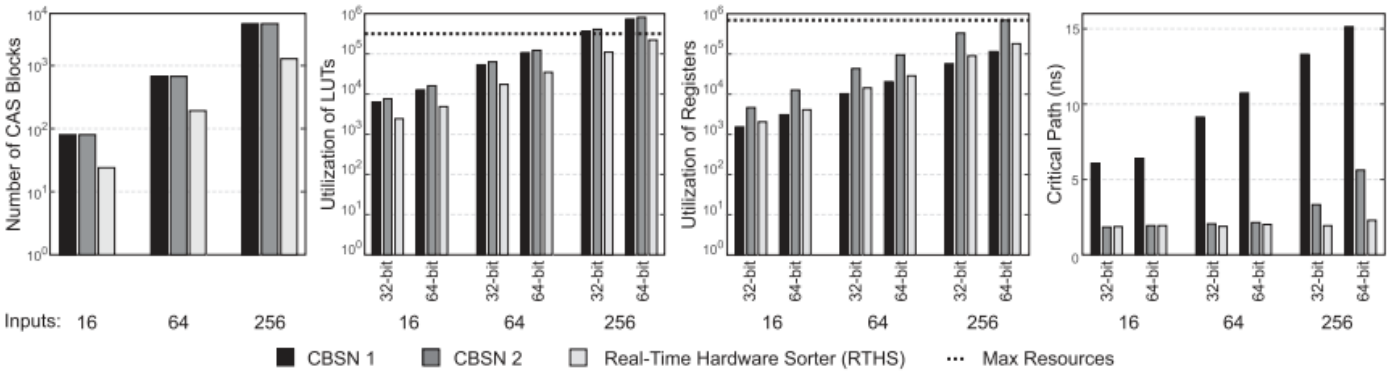
در جدول ۲، تعداد LUTها به طور متوسط ٪۶۶٫۳ در مقایسه با CBSN1 کاهش یافته است. برخلاف LUTها، تعداد رجیسترها به طور متوسط ٪۴۳٫۵ افزایش یافته است. همان‌طور که انتظار می‌رفت، دلیل اصلی این افزایش، تعداد رجیسترهای پایپلاین استفاده‌شده در طراحی RTHS است.

مقیاس‌پذیری (Scalability) یکی از مهم‌ترین مسائل در مسئله مرتب‌سازی است. طبق ارزیابی‌های انجام‌شده، مقیاس‌پذیری طراحی RTHS بسیار برتر از CBSN است. با افزایش تعداد ورودی‌ها، تأثیر کمی بر مسیر بحرانی (critical path) و افزایش آهسته‌ای در نیازهای منابع طراحی مشاهده می‌شود، که این عوامل به طور قابل توجهی به مقیاس‌پذیری RTHS کمک می‌کنند. جدول ۳ و شکل ۱۱ طراحی RTHSرا در مقایسه باPMT ، SHMS، RSNو unary design برای ۱۶، ۳۲، ۶۴ و ۲۵۶ رکورد ورودی بررسی می‌کنند. پیاده‌سازی PMT و SHMS برای ۲۵۶ رکورد ورودی به دلیل پیچیدگی بالا و نیازهای منابع انجام نشده است، بنابراین در این دو مورد، نتایج به‌صورت فرضی تخمین زده شده‌اند. unary designs نیاز به کمترین میزان منابع برای پیاده‌سازی دارد.

تعدادLUT های استفاده‌شده توسط RTHS در مقایسه با طراحی‌های PMTو SHMS به ترتیب ٪۸۵٫۸ و ٪۸۷٫۳ کاهش یافته و در مقایسه با طراحی یگانه بین ٪۲۹٫۹ و ۱٫۳ برابر افزایش یافته است. مشابه LUTها، ما در تعداد رجیسترهای استفاده‌شده به ترتیب ٪۶۵٫۸ و ٪۹۴٫۸ صرفه‌جویی نسبت به طراحی‌های PMT و SHMS را به دست

جدول ۳: مقایسه طراحی RTHS و طراحی‌های یوناری، PMT، SHMS، وRSN

# of I/O	# of DPBN	# of I/O in DPBN	CAS blocks			Pipeline stages			Data width	LUTs			Registers			Critical path (ns)			Frequency (MHz)		
			CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS		CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS
16	4	4	80	80	24	4	10	4	32-bit	*6,344 (2%)*	*7,058 (2.3%)*	*2,437 (0.8%)*	*1,536 (0.25%)*	*5,632 (0.93%)*	*2,051 (0.33%)*	*6.061*	*1.822*	*1.871*	*151.2*	*548.7*	*534.4*
									64-bit	*12,744 (4%)*	*14,098 (4.7%)*	*4,869 (1.6%)*	*3,072 (0.5%)*	*11,264 (1.8%)*	*4,099 (0.6%)*	*6.397*	*1.942*	*1.936*	*156.3*	*514.8*	*516.4*
64	8	8	672	672	192	6	21	6	32-bit	*53,376 (17%)*	*59,728 (20%)*	*17,413 (5%)*	*10,240 (1%)*	*45,060 (7.4%)*	*14,340 (2%)*	*9.138*	*1.923*	*1.886*	*109.4*	*520*	*530.2*
									64-bit	*107,136 (35%)*	*113,040 (37%)*	*34,823 (11%)*	*20,480 (3%)*	*90,112 (15%)*	*28,675 (4%)*	*10.27*	*2.025*	*2.013*	*97.3*	*493.8*	*496.7*
256	16	16	4,608	4,608	1,280	8	36	10	32-bit	*365,696 (120%)*	*379,340 (125%)*	*110,605 (36%)*	*57,344 (9%)*	*303,104 (50%)*	*90,115 (14%)*	*13.299*	*3.193*	*1.932*	*75.2*	*313.2*	*517.6*
									64-bit	*736,128 (242%)*	*758,339 (250%)*	*221,197 (72%)*	*114,688 (18%)*	*606,208 (99%)*	*180,227 (29%)*	*15.138*	*5.631*	*2.289*	*66*	*177.5*	*436.8*



شکل ۱۱: تعداد LUTها، ثبات‌ها، و مسیر بحرانی (نانومتر) گزارش شده برای طراحی‌های unary، PMT، SHMS، RSN، و RTHS.

آمده است. طول مسیر بحرانی (critical path) در طراحی RTHS بسیار کمتر از سایر روش‌های مرتب‌سازی است و مقدار نسبتاً ثابتی را ارائه می‌دهد. مسیر بحرانی (critical path) در RTHS به یک بلوک CAS دو حالته وابسته است، در حالی که در سایر روش‌های مرتب‌سازی برابر با حداکثر تعداد بلوک‌های CAS متوالی است. اگر تأخیر سوئیچ ضمنی را ثابت فرض کنیم، تأخیر کلی RTHS با افزایش تعداد رکوردهای ورودی افزایش نمی‌یابد. با این حال، در واقعیت، پیچیدگی سوئیچ منجر به مسیریابی پیچیده‌تر در FPGA می‌شود که به نوبه خود مسیر بحرانی (critical path) را به تدریج افزایش می‌دهد. مسیر بحرانی (critical path) طراحی مقاله بررسی شده به ترتیب ۸۰,۳٪، ۴۳,۲۵٪ و ۲۶,۵٪ کوتاه‌تر از طراحی‌های PMT، SHMS و unary design، است. بنابراین، بالاترین فرکانس را در میان سایر طراحی‌های مرتب‌سازی به دست آمده است.

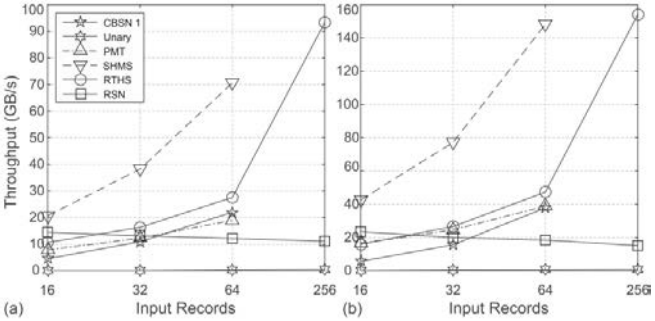
تحلیل عملکرد

بازدهی یک مرتب‌کننده با N رکورد ورودی بر اساس فرمول زیر تعیین می‌شود:

$$T = N/t \times f \times data_{width} \quad (۱۱)$$

که در آن f فرکانس، t زمان مورد نیاز برای مرتب‌سازی N رکورد ورودی (بر حسب سیکل‌ها) و پهنای باند داده‌ها بر حسب بایت است. در طراحی RTHS، از N/2بلوک حافظه RAM دو پورتی (BRAM) به همراه یک واحد کنترل BRAM برای ذخیره مراحل خط لوله و رکوردهای ورودی از حافظه اصلی استفاده می‌شود. بهره‌گیری از BRAMها به کاهش تأخیرهای خواندن و نوشتن مستقیم رکوردها از حافظه کمک می‌کند که موجب افزایش بازدهی مرتب‌کننده می‌شود. FPGA مدل Virtex-7 با قابلیت استفاده از BRAMهای دو پورتی، امکان خواندن و نوشتن همزمان را فراهم می‌کند. تعداد

رکوردها از ظرفیت مرتب‌سازی RTHS بیشتر خواهد بود، لذا BRAMها هر مجموعه از رکوردهای ورودی را به‌طور سریالی در دو سیکل ساعت به رجیسترهای مرتب‌سازی منتقل می‌کنند. پس از اتمام انتقال، فرایند مرتب‌سازی برای هر مجموعه رکوردها آغاز می‌شود. با استفاده از الگوریتم بایتونیک نرم‌افزاری، رکوردها یکی پس از دیگری به مرتب‌ساز تغذیه می‌شوند. هر BRAM به دو بخش تقسیم می‌شود؛ بخش دوم برای ذخیره‌سازی توالی بعدی رکوردهای ورودی استفاده می‌شود. بنابراین، در حالی که مرتب‌ساز در حال مرتب‌سازی رکوردهای بخش اول است، رکوردهای بخش دوم آماده می‌شوند.

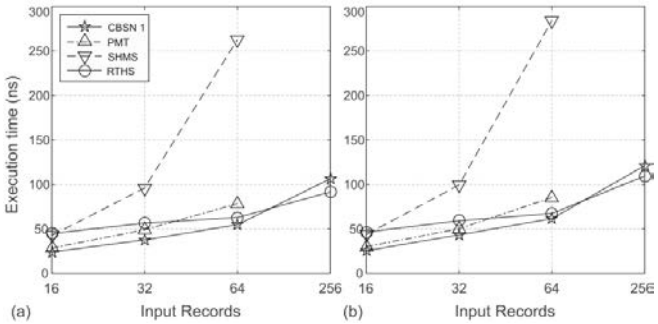


شکل ۱۲: نرخ عبور روش‌های مرتب‌سازی برای تعداد مختلف رکوردهای ورودی. (الف) عرض داده ۳۲ بیت. (ب) عرض داده ۶۴ بیت.

شکل ۱۲ بازدهی طراحی RTHS را در مقایسه با روش‌های unary، PMT و SHMS نشان می‌دهد. برای طراحی‌های PMT و SHMS با ۲۵۶ ورودی، بازدهی مشخص نشده است زیرا امکان پیاده‌سازی وجود ندارد. طراحی unary برای مرتب‌سازی NNN رکورد با عرض MMM بیت، به M۲ سیکل نیاز دارد که بسیار طولانی‌تر از تأخیر CBSN (طولانی‌ترین مسیر بحرانی) است. اگر عرض داده کم باشد، طراحی unary می‌تواند مقرون‌به‌صرفه باشد، اما با عرض‌های داده ۳۲ بیتی و ۶۴ بیتی، کمترین بازدهی را خواهد داشت. در نظر داشته باشید که طراحی RTHS ابتدا یک دنباله کاملاً نامرتب از رکوردها را دریافت می‌کند، در حالی که روش‌های PMT و SHMS دنباله‌هایی از رکوردهای جزئی مرتب شده را به عنوان ورودی دریافت می‌کنند و یک دنباله مرتب شده را خروجی می‌دهند. ما در این مقایسه، سربار عملیات مرتب‌سازی جزئی اولیه برای روش‌های مذکور را در نظر نگرفته‌ایم.

تحلیل زمان اجرا

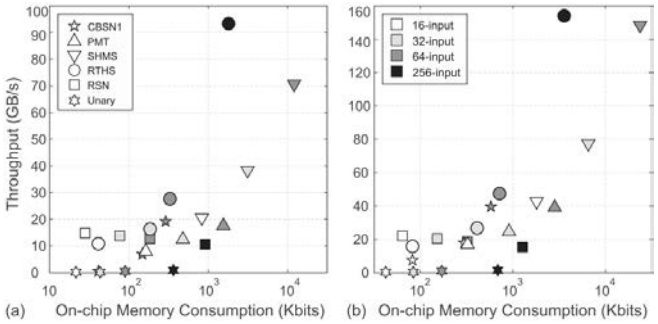
در شکل ۱۳، زمان اجرای مختلف روش‌های مرتب‌سازی بر اساس تعداد رکوردهای ورودی نمایش داده شده است. در این آزمایش، در ابتدای هر مرحله، تنها یک دنباله رکورد به هر یک از الگوریتم‌های مرتب‌سازی داده می‌شود و زمان اجرا به تفاوت بین زمان ارسال رکوردها به الگوریتم و زمان دریافت رکوردهای مرتب‌شده اندازه‌گیری می‌شود. همان‌طور که انتظار می‌رفت، روش RTHS از نظر زمان اجرا نسبت به دیگر روش‌ها پایداری بیشتری نشان می‌دهد. باید توجه داشت که تکنیک پایپ‌لاین در شکل ۱۳ تأثیرگذار نیست، چرا که تنها یک دنباله رکورد به الگوریتم‌ها ارائه شده است. همچنین، استفاده از فرآیند سریالی در روش SHMS به‌طور قابل توجهی بر عملکرد آن تأثیرگذار است. به دلیل تأخیرات زیاد روش Unary در این سناریو، امتیازهای آن در شکل ۱۳ درج نشده است. علاوه بر این، زمان اجرای RSN در شکل ۱۳ نشان داده نشده است، زیرا این زمان به نرم‌افزار مرتب‌سازی بستگی داشته و معمولاً از دیگر روش‌ها بیشتر است.



شکل ۱۳: نرخ عبور روش‌های مرتب‌سازی برای تعداد مختلف رکوردهای ورودی. (الف) عرض داده ۳۲ بیت. (ب) عرض داده ۶۴ بیت.

ارزیابی کارایی حافظه

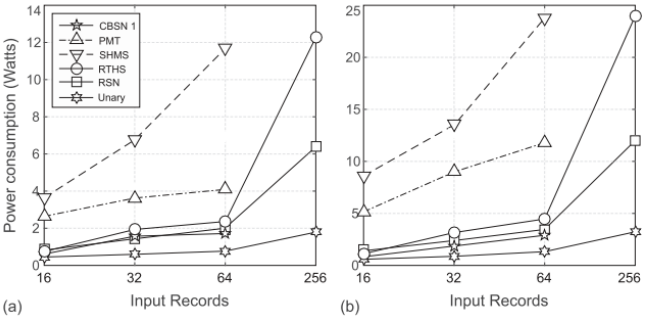
در شکل ۱۴، کارایی حافظه پنج روش مختلف مرتب‌سازی با یکدیگر مقایسه شده است. برای یک الگوریتم مرتب‌سازی ایده‌آل، هدف این است که با حداقل مصرف حافظه، بالاترین بازدهی را به دست آورد. در این مقایسه، روش SHMS که از نظر بازدهی برتری نسبت به RTHS دارد، در سمت راست نمودار کارایی حافظه قرار دارد، زیرا این روش به حافظه بیشتری نیاز دارد. در عوض، روش RTHS با استفاده از ۲۵۶ رکورد و عرض ۶۴ بیت، در قسمت بالا سمت چپ نمودار بهینه‌ترین استفاده از حافظه را ارائه می‌دهد و از نظر کارایی حافظه در بین روش‌های دیگر برتر است.



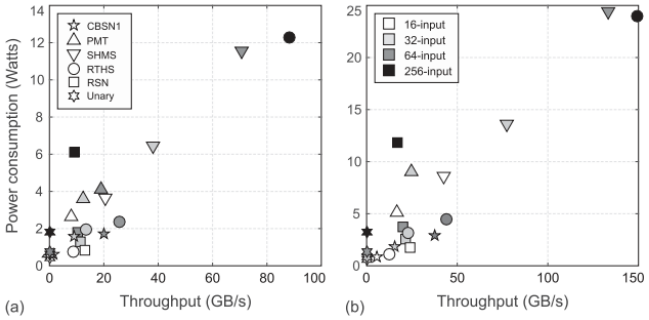
شکل ۱۴: کارآیی حافظه بین پنج روش مرتب‌سازی برای تعداد مختلف رکوردهای ورودی. (الف) عرض داده ۳۲ بیت. (ب) عرض داده ۶۴ بیت.

بررسی مصرف توان

در شکل ۱۵، مصرف توان روش پیشنهادی در مقایسه با دیگر روش‌های مرتب‌سازی نمایش داده شده است. مصرف توان یکی از جنبه‌های حیاتی در طراحی سیستم‌های دیجیتال به حساب می‌آید و در تراشه‌های FPGA به طراحی، شرایط عملیاتی، دستگاه هدف و فرکانس کاری وابسته است. علاوه بر این، مصرف توان تحت تأثیر عواملی چون استفاده از منابع، محل‌یابی و مسیریابی، نقشه‌برداری و تقسیم‌بندی منطق قرار دارد. مصرف کل توان در تراشه‌های FPGA از مصرف توان ایستا و پویا حاصل می‌شود و شامل توان سیگنال ساعت، سیگنال‌های طراحی، بلوک‌های منطقی و ورودی/خروجی است. همان‌طور که در شکل ۱۵ مشاهده می‌شود، با افزایش تعداد رکوردهای ورودی، مصرف توان تمامی روش‌ها افزایش می‌یابد. روش Unary به دلیل کاهش شدید در منابع مورد نیاز، کمترین مصرف توان را دارد. در مقابل، روش SHMS با توجه به نیازهای بالای منابع و فرکانس عملیاتی، بالاترین مصرف توان را دارد. در کل، روش RTHS به نسبت بیشتر از CBSN1 و Unary به انرژی نیاز دارد. در حالی که فرکانس بالای عملیاتی و وابستگی به یک کنترلر باعث افزایش مصرف توان در روش پیشنهادی شده، این روش همچنان نسبت به روش‌های PMT و SHMS مصرف توان کمتری دارد به دلیل کاهش قابل توجه در مصرف منابع. برای مقایسه عادلانه‌تر، نسبت توان به خروجی می‌تواند معیار بهتری باشد و شکل ۱۶ این نسبت را برای روش RTHS در مقایسه با سایر روش‌های مرتب‌سازی نمایش می‌دهد. نتایج قابل قبول در زیر محور y = x قرار دارند. به عبارت دیگر، پایین و راست پایین نمودار نماینده طراحی ایده‌آل‌تر است که به مصرف کمتر توان همراه با کارایی بالاتر منجر می‌شود. روش پیشنهادی RTHS همیشه زیر محور y = x قرار دارد به جز برای پیاده‌سازی با ۳۲ رکورد و عرض داده ۳۲ بیت.



شکل ۱۵: مصرف توان روش‌های مرتب‌سازی برای تعداد مختلف رکوردهای ورودی. (الف) عرض داده ۳۲ بیت. (ب) عرض داده ۶۴ بیت.



شکل ۱۶: نسبت نرخ عبور به مصرف توان پنج روش مرتب‌سازی. (الف) عرض داده ۳۲ بیت. (ب) عرض داده ۶۴ بیت.

۷- جمع‌بندی و نتیجه‌گیری

مرتب‌سازی به عنوان یکی از وظایف اساسی در بسیاری از سیستم‌ها و برنامه‌ها اهمیت دارد. FPGAها به دلیل کارایی بالا و بهینه بودن از نظر حافظه، گزینه‌ای مناسب برای پیاده‌سازی طراحی‌های موازی و لوله‌ای در معماری‌های مختلف هستند. در این مقاله، الگوریتم جدیدی به نام MDSA و RTHS ارائه شده که از نظر عملکرد بسیار مؤثر است. این الگوریتم به تنها شش مرحله برای مرتب‌سازی هر تعداد رکورد نیاز دارد. ما یک تحلیل جامع از RTHS در زمینه‌های مختلفی مانند استفاده از منابع، زمان اجرا، حافظه و توان عملیاتی انجام داده‌ایم. نتایج تحلیل‌ها نشان می‌دهد که RTHS به طور قابل توجهی مصرف منابع را کاهش می‌دهد و تعداد LUTs نسبت به روش CBSN به میزان ۶۶٫۳٪ کاهش یافته است. همچنین، RTHS موفق شده است تا در مقایسه با روش پیشرفته SHMS، تعداد LUTs و registers را به ترتیب ۸۷٫۳٪ و ۹۴٫۸٪ کاهش دهد. در برنامه‌های آینده، قصد داریم صف‌های Min/Max بزرگ را برای زمان‌بندی وظایف در سیستم‌های زمان واقعی و روی پلتفرم‌های چند هسته‌ای پیاده‌سازی کنیم.