علوم رایانش و فناوری اطلاعات
نشریه علمی انجمن کامپیوتر ایران
مجلد ۱۹، شماره ۱، ۱۴۰۰
صفحات ۱۴- ۲۲
مقاله عادی

بهبود متوسط زمان پاسخگویی حافظه برای برنامه‌های حافظه-محور به منظور کاهش زمان بیکاری هسته‌های پردازشی در پردازنده گرافیکی

حسین بی طالبی[۱]، فرشاد صفایی[۲*]

*نویسنده مسئول، دریافت: ۱۴۰۰/۰۲/۱۹، بازنگری: ۱۴۰۰/۰۲/۲۵، پذیرش: ۱۴۰۰/۰۲/۲۹

[۱] دانشجوی دکتری، دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی، تهران، ایران
[۲] دانشیار، دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی، تهران، ایران

چکیده

ظهور مفهوم GPGPU همراه با CUDA ومدل‌های برنامه نویسی نظیر OpenCl، فرصت‌های جدیدی را برای کاهش تأخیر و توان مصرفیِ برنامه‌های کارایی محور فراهم می‌کند. GPU می‌تواند هزاران نخ پردازشی موازی را برای پنهان کردن تأخیر پرهزینه دسترسی به حافظه اجرا کند. با این حال، برای برخی از برنامه‌های حافظه محور، به احتمال زیاد در برخی فواصل زمانی تمام نخ‌های پردازشی یک هسته متوقف شده و منتظر تأمین داده توسط واحد حافظه هستند. در این پژوهش هدف ما بهبود تأخیر دسترسی به حافظه برای بسته‌های تولیدی توسط هسته‌های بحرانی در پردازنده‌های گرافیکی است. به منظور بهبود زمان غیربهینه هسته‌ها، ما بر روی شبکه میان ارتباطی بین هسته‌ها و حافظه پنهان سطح آخر تمرکز و بسته مربوط به هسته‌هایی که تعداد بیشتری نخ متوقف شده دارند را در ورود به شبکه و داوری در شبکه اولویت قرار می‌دهیم. به این ترتیب ، بیشترین اولویت در داوری و تخصیص منابع به بسته‌های بحرانی‌تر اعطا می‌شود، بنابراین درخواست حافظه برای آنها سریعتر سرویس دهی شده و متوسط زمان توقف هسته کاهش و در نهایت کارایی پردازنده گرافیکی افزایش می‌یابد.

کلمات کلیدی: پردازنده گرافیکی، شبکه میان ارتباطی، تاخیر، سطوح اولویت، بحرانی، حافظه، حالت بیکاری، حافظه‌نهان.

۱- مقدمه

واحد پردازش گرافیکی همه منظوره (GPGPU) ، به عنوان جایگزینی کارا و مقرون به صرفه برای واحد پردازش مرکزی (CPU) برای طیف وسیعی از برنامه‌های موازی و توان محور در نظر گرفته می‌شوند [۱-۶]. GPUها به لطف داشتن تعداد بسیار زیاد چندپردازنده جریانی(SM) هزاران نخ فعال را به صورت موازی به اجرا دراورند. برای پشتیبانی از چنین سطح عظیمی از موازی سازی، چندین مدل برنامه نویسی سفارشی مانند OpenCL [۷] و CUDA [۸] ساخته شده است. برنامه GPGPUها از چندین هسته تشکیل شده است، هر یک از آنها حاوی تعداد زیادی نخ پردازشی است. در GPGPU ، نخ‌ها به عناصر مختلف به اصطلاح بلوک‌های نخی تقسیم می‌شوند (همچنین به عنوان Cooperative Thread Array - CTA هم شناخته می‌شوند). هنگامی که یک برنامه موازی در GPGPU شروع به کار می‌کند ، یک واحد زمانبندی بلوک نخ، بلوک‌های نخ را به SMها اختصاص می‌دهد به گونه ای که تمام رشته‌های داخل یک بلوک نخ به یک SM اختصاص می‌یابد. در داخل SMها ، هر بلوک نخ به زیر گروههای کوچکتر متشکل از ۳۲ نخ تقسیم می‌شود که Warp نامیده می‌شوند. همه رشته‌های یک Warp یک دستورالعمل یکسان را برای تحقق بخشیدن به مدل پردازش "SIMT" اجرا می‌کنند [۹،۱۰]. به این ترتیب GPGPUها قادر به پردازش همزمان هزاران نخ و دستیابی به سرعت قابل توجه هستند [۱۱]. با همه این موارد به دلیل برخی محدودیت‌های معماری و حافظه ، هسته‌های پردازشی بازه‌هایی از عدم فعال بودن را تجربه می‌کنند بنابراین به سختی می‌توان به حداکثر عملکرد آنها دست یافت. به صورت کلی سه دلیل مهم برای این محدودیت می‌توان بیان نمود [۶]. دلیل اول محدودیت در سایز حافظه‌های روی تراشه و رجیستر فایل داخلی است که تاثیر بسزایی در میزان موازی سازی دارد. به عنوان مورد دوم، وجود واگرایی بین نخ‌های یک warp است که تمام نخ‌های پردازشی را تا زمان همگرایی در یک نقطه کنترل مشخص متوقف می‌کند، و سوم توقف فعالیت SM به دلیل تاخیر دسترسی به حافظه است. درخواست حافظه صادر شده از SM باید از چندین سطح از سلسله مراتب حافظه عبور کند. در طول این مسیر ، شبکه میان ارتباطی بین LLC و کنترل کننده‌های حافظه یکی از طولانی ترین تأخیرها را دارد. از طریق شبکه میان ارتباطی، درخواست‌های تولید شده توسط

تمام SMها به کنترل کننده حافظه (MCs) هدایت می‌شوند. به دلیل معماری پردازنده‌های گرافیکی که تعداد واحدهای کنترل کننده حافظه کمتر از تعداد SMها است، ساختار ترافیکی Many-to-Few-to-Many در طی مسیریابی در شبکه شکل میگیرد [۱۲].

برنامه‌ها را می‌توان به دو دسته کلی حافظه محور و محاسبه محور تقسیم کرد. برنامه‌های محاسبه محور درخواست حافظه کمتری دارند و زمان بیشتری برای پردازش داده‌ها مصرف می‌کنند، در حالی که برنامه‌های حافظه محور بیشتر زمان اجرا را صرف واکشی داده‌ها از سلسله مراتب حافظه می‌کنند [۱۴،۱۳]. در یک GPU معمولی زمان‌بند، warp مربوط به نخ فعال فعلی را که در دسترسی به حافظه‌نهان سطح یک ناکام مانده است را با یک warp آماده برای اجرا جایگزین می‌کند. نخ‌هایی که منتظر داده هستند تا دریافت پاسخ از حافظه در صف پاسخگویی قرار می‌گیرند. اگر میزان warpهای متوقف شده در صف پاسخگویی حافظه به قدری افزایش یابد که دیگر هیچ warp اماده ای در صف باقی نماند در این صورت SM مربوطه به ناچار به حالت بیکاری فرو می‌رود [۱۶،۱۵]. در برخی از برنامه‌های حافظه محور ممکن است برخی از SMها در طول اجرای برنامه به شرایط بحرانی برسند که بدلیل تاخیر پاسخگویی حافظه warp اماده ای برای اجرا نداشته باشند و برخی از SMهای دیگر همچنان به دلیل داشتن warp قادر به اجرای دستورات باشند. با این حال ، SMهایی که کمتر دچار بحران warp اماده هستند، تاخیر دسترسی به حافظه برای آنها چندان مهم نیست. به عبارت دیگر، برخی از SMها تحمل پذیری بالاتری نسبت به تاخیر دسترسی به حافظه دارند (و تاخیر شبکه) ، زیرا می‌توانند این تاخیر را توسط تعداد زیاد warp آماده برای اجرا پنهان کنند. در واقع اگر به بسته‌های تولید شده توسط SMهای بحرانی تر در ورود به شبکه و همچنین در حین مسیریابی در شبکه اولویت بالاتری اعمال نماییم قادر خواهیم بود نرخ بیکاری SMها را به ازای برنامه‌های حافظه محور مانند برنامه‌های شبکه عصبی کاهش داده و در نتیجه کارایی را بهبود چشمگیری بخشیم.

ادامه مقاله به شرح زیر است: بخش ۲ پیش‌زمینه مرتبط با این پژوهش را بیان می‌کند. انگیزه‌های اصلی انجام پژوهش در بخش ۳ مطرح شده است. رویکردهای پیشنهادی تحقیق ما در بخش ۴ بیان شده است. بخش ۵ معیار و ویژگی‌های شبیه ساز را معرفی می‌کند. نتایج طرح‌های پیشنهادی در مورد عملکرد GPU و میانگین تأخیر دسترسی حافظه در بخش ۶ ارزیابی شده است. در بخش ۷ کارهای مرتبط با پژوهش جاری را بررسی کردیم. و به عنوان آخرین بخش، نتیجه گیری پژوهش خود را در بخش ۸ انجام داده ایم.

۲- پیش‌زمینه

در این بخش ما نگاهی مختصر به معماری کلی پردازنده گرافیکی خواهیم داشت و نشان می‌دهیم هنگامی که یک درخواست حافظه ایجاد می‌شود تا پاسخ ان دریافت گردد با چه موانعی بر سر راه خود مواجه خواهد شد. شبکه میان ارتباطی و مشکلات ان‌ها را در پردازنده گرافیکی مطرح می‌نماییم. علل کاهش کارایی کش سطوح مختلف و حافظه را بیان می‌کنیم و در بخش بعدی انگیزه اصلی طرح مساله‌ی این مقاله بیان خواهد شد.

۲-۱- ذخیره سازی و حافظه

در اولین سطح از سلسله مراتب حافظه GPU حافظه نهان L1 در نزدیک حالت به SMها قرار دارد. هر SM حافظه پنهان L1 اختصاصی خود را دارد، وقتی آدرس درخواستی در خطوط حافظه پنهان L1 در دسترس نباشد، درخواست حافظه نهان سطوح پایین‌تر که معمولاً در میان SMهای مختلف مشترک است ارجاع می‌شود. در نتیجه اگر در هیچ یک از حافظه‌های پنهان داده مناسبی وجود نداشته باشد، حافظه اصلی به درخواست‌ها پاسخ می‌دهد. واحدهای کنترل کننده حافظه اصلی درخواست‌های حافظه را که از SMهای مختلف دریافت می‌شود، زمان بندی و مدیریت می‌کنند و براساس الگوریتم خاصی که Round Robin یکی از پرکاربردترین آنها است درخواست‌ها را پاسخ میدهند [۱۷]. پردازنده‌ها به طور معمول از سطوح مختلف حافظه نهان پشتیبانی می‌کنند تا هزینه‌های دسترسی به حافظه اصلی را کاهش دهند. حافظه پنهان L1 به عنوان سریعترین و کوچکترین قسمت سلسله مراتب حافظه است که تقریباً با سرعتی نزدیک به سرعت پردازنده داده‌ها را برای SMها فراهم میکند. سطوح پایین‌تر حافظه نهان معمولاً فضای ذخیره سازی بیشتری نسبت به سطح بالاتر دارند اما سرعت آماده سازی اطلاعات آنها بسیار کندتر است.

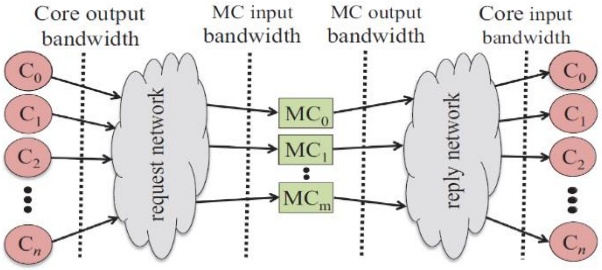
۲-۲- زمان‌بند دستورالعمل

برنامه‌هایی که برای اجرا به پردازنده گرافیکی اختصاص داده می‌شوند به قسمت‌های مختلفی تقسیم می‌شوند. یک بلوک از نخ‌های پردازشی شامل تعدادی نخ همگام است که تحت عنوان CTA شناخته میشوند. الگوریتم‌های مختلفی برای زمان بندی CTA وجود دارد [۱۹،۱۸]، یک زمان‌بند بهینه می‌تواند عملکرد پردازنده گرافیکی را تا حد زیادی بهبود بخشد و زمان بیکاری هسته‌ها را کاهش دهد. warp کوچکترین بخش این تقسیم‌بندی است که به تعداد Streaming Processor(SP)های موجود در SM دارای نخ پردازشی می‌باشد. هر warp ممکن است درحین اجرای برنامه چندین سیکل را به دلیل محدودیت منابع و کندی پاسخگویی حافظه در حالت بلوکه قرار بگیرد. زمانبند ممکن است چندین Warp را به یک SM اختصاص دهد ، وقتی اجرای هر یک از آنها به پایان رسید ، یک warp آماده از صف آماده جایگزین می‌شود. Warpها را می‌توان از نظر موقعیت در سه گروه اصلی دسته بندی کرد. (I) دسته اول شامل Warpهایی است که SM را در اختیار دارد و در حال اجرا است. (II) دسته دوم شامل warpهایی است که به دلیل کمبود منابع مناسب مسدود شده و منتظر آن هستند. (۳) گروه سوم نیز Warpهایی هستند که برای دریافت SM در صف آماده منتظر می‌مانند. GPU به دلیل ماهیت اجرای چندین نخ پردازشی در هر SM مفهوم Single-Instruction-Multiple-Thread (SIMT) را اجرا می‌کنند. طول SIMT با تعداد نخ‌های مستقل در حال اجرا در یک SM مشخص می‌شود.

۲-۳- شبکه میان‌ارتباطی

شبکه میان ارتباطی در پردازنده‌گرافیکی وظیفه ان را دارد تا درخواست حافظه را از SMها به کش سطح پایین‌تر یا حافظه برساند و پاسخ را به SM درخواست دهنده بازگرداند. با توجه به انچه که در شکل ۱ امده است در پردازنده‌های گرافیکی تعداد زیادی هسته پردازشی درخواست‌های حافظه خود را به شبکه میان‌ارتباطی وارد می‌کنند. درخواست‌ها پس از مسیریابی، از گره‌های کنترل‌کننده حافظه که تعداد ان‌ها بسیار کمتر از هسته‌های-پردازشی است، از شبکه خارج می‌شوند. پس از اماده شدن درخواست، حجم بسیار بالای پاسخ‌ها از گره‌های کنترل‌کننده حافظه که تعداد ان‌ها به نسبت هسته‌های-پردازشی بسیار کمتر است وارد شبکه پاسخ می‌شوند. تعداد SMها بسیار بیشتر از تعداد MCها است و این موضوع باعث می‌شود که شبکه میان ارتباطی در پردازنده‌گرافیکی تعادل ترافیکی نداشته باشد. در نزدیکی گره‌های MC، ترافیک شدید می‌شود و بسته‌ها در روترهای نزدیک به MC زمان بیشتری را در بافر صرف می‌کنند تا پورت مناسب را در اختیار بگیرند. همچنین به دلیل بالا بودن تراکم شبکه در نزدیکی MCها میزان نزاع در شبکه افزایش می‌یابد. در واقع MCها نقاط بحرانی در شبکه میان‌ارتباطی پردازنده گرافیکی هستند و یکی از گلوگاه‌های اصلی کارایی سیستم به شمار می‌روند [۲۱،۲۰].

در طول اجرای برنامه تعداد درخواست‌های خواندن از حافظه (Read) بسیار بیشتر از نوشتن در حافظه است. بسته‌های Read تنها شامل ادرس مشخصی از حافظه و بیت‌های کنترل هستند. اما پاسخ ان‌ها علاوه بر ادرس مشخص و بیت‌های کنترل، شامل دادهای بلاک ادرس درخواست می‌باشد، بنابراین سایز بسته پاسخ به درخواست Read بسیار بزرگتر از خود درخواست Read می‌باشد. و این موضوع برای درخواست Write دقیقا برعکس است در نتیجه ترافیک شبکه پاسخ بسیار بیشتر از شبکه درخواست می‌باشد. به دلیل رفتار و خاصیت ترافیکی پردازنده گرافیکی و به منظور جلوگیری از بن‌بست و از کار افتادن شبکه، دو شبکه میان‌ارتباطی مجزای درخواست و پاسخ در پردازنده گرافیکی وجود دارد. در شبکه درخواست تنها بسته‌های تولید شده از SMها تا MC مسیر یابی می‌شوند و در شبکه پاسخ نیز تنها بسته‌های اماده شده ازMC به سمت SM مسیریابی می‌شوند.



شکل ۱: نمایش مدل ترافیکی Many-to-Few-to-Many در شبکه میان ارتباطی پردازنده گرافیکی

۲-۴- محدودیت منابع

اجرای همزمان تعداد زیادی از نخ‌های پردازشی ویژگی مهمی است که در سال‌های اخیر پردازنده‌های گرافیکی را از پردازنده‌های دیگر پیش انداخته است. متأسفانه محدودیت‌های منابع مانع از آن می‌شود که GPUها به حداکثر ظرفیت پردازش برای اجرای نخ‌های همزمان برسند. تأخیر طولانی آماده‌سازی داده‌های حافظه اصلی، محدودیت سطح مختلف حافظه پنهان در مقدار سایز ذخیره سازی آنها، سایز کوچک بافر درخواست و محدودیت سایز بافرهای شبکه از دلایلی است که باعث افزایش زمان پاسخ دسترسی به حافظه می‌شود. این محدودیت‌ها به دلایل مختلف مانند سربار توان و مساحت اشغالی با تکنولوژی‌های فعلی به طور مستقیم قابل حل نیستند. در این تحقیق، ما روش جدیدی برای کاهش تأثیر محدودیت منابع بر عملکرد پردازنده‌های گرافیکی ارائه می‌دهیم. شبکه میان‌ارتباطی به عنوان یکی از مهمترین بخش‌های پردازنده‌های گرافیکی، می‌تواند نقش بسزایی در عملکرد و قابلیت اطمینان مجموعه داشته باشد. دسترسی به حافظه یکی از مهمترین عوامل در عملکرد پردازنده است. تأخیر شبکه در بسیاری از برنامه‌های حافظه محور نقش بسیار مهمی در تأخیر دسترسی به حافظه دارد. با این وجود پژوهش‌ها در زمینه شبکه میان‌ارتباطی GPU بسیار ناچیز است. در حالی که می‌توان با درک رفتار شبکه ارتباطی پیشرفت چشمگیری در عملکرد و هزینه GPU ایجاد کرد. به منظور درک بهتر محدودیت منابع در این تحقیق از برنامه محک‌های مختلفی برای ارائه مطالعات دقیق‌تر و جامع‌تر استفاده شده است. این معیارها به سه دسته برنامه‌های پردازش محور ، برنامه‌های حافظه محور و برنامه‌های خنثی طبقه بندی می‌شوند.

۳- انگیزه پژوهش

حافظه بخشی جدایی ناپذیر از هر واحد محاسباتی و پردازشی است. هر واحد پردازش توسط حافظه تغذیه می‌شود و خروجی را به آن باز می‌گرداند. اولین قدم قبل از شروع اجرای برنامه توسط واحدهای پردازشی، واکشی دستورالعمل‌ها برای تعیین نوع آنها می‌باشد. بعد از واکشی ان‌ها از حافظه، واحد رمزگشایی نوع دستورالعمل را مشخص می‌کند و سپس داده‌های مناسب را از حافظه درخواست می‌دهد. درنهایت پس از پایان اجرای دستورالعمل خروجی‌ها در واحد حافظه بازنویسی می‌شوند. همانطور که مشخص است، حافظه تأثیرات زیادی بر عملکرد هر واحد پردازنده دارد. با وجود بهبود قابل توجه زمان دسترسی به حافظه در سال‌های اخیر همچنان زمان دسترسی به حافظه به عنوان گلوگاه اصلی عملکرد سیستم‌های پردازشی شناخته می‌شود. GPUها از چالش‌های حافظه مستثنی نیستند و با وجود قدرت پردازشی بالا به دلیل محدود بودن پهنای باند و مدت زمان دسترسی حافظه قادر به دستیابی به حداکثر عملکرد خود نخواهند بود. کاهش عملکرد یکی از چالش‌های اصلی پردازنده‌های گرافیکی به دلیل کندی سرعت دسترسی به حافظه است. بسیاری از تحقیقات طرح‌های مختلفی را برای کاهش هزینه‌های اضافی دسترسی حافظه بر روی عملکرد GPUها ارائه داده‌اند، با این وجود، تأخیر دسترسی به حافظه همچنان گلوگاه اصلی برای دستیابی به عملکرد نهایی GPUها است. مطالعات در مورد تأثیر تأخیر حافظه اصلی بر عملکرد GPU دارای طیف گسترده ای است که در بخشهای مختلف ساختار GPU وجود دارد. بهبود در واحد کنترل کننده حافظه [۲۳،۲۲]، واحد واکشی [۲۴]، واحد شاخه و پرش [۲۵]، واحد زمانبند CTA ، واحد زمانبند Warp [۲۶] از جمله این تحقیقات هستند. بدین منظور، ما یک روش موثر و کارا برای کاهش میانگین زمان پاسخ حافظه برای SMهای بحرانی که نزدیک به حالت بیکاری هستند، پیشنهاد کرده ایم.

هنگامی که warp یک SM را در اختیار میگیرد، ممکن است در زمان اجرایش چندین درخواست حافظه داشته باشد. با تولید هر درخواست حافظه، این warp بلوکه می‌شود و با یک warp آماده جایگزین می‌گردد. Warp بلوکه شده تا زمانی که پاسخ مناسب از حافظه دریافت کند در این وضعیت باقی خواهد ماند. وضعیت درخواست‌های ناموفق به حافظه نهان L1 در Miss Status Holding Register(MSHR) قرار می‌گیرد و پس از فرآیند ادغام درخواست متناظر به شبکه تزریق می‌شود. بسته‌های درخواستی از عناصر مختلف شبکه عبور می‌کنند (به عنوان مثال روترها و بافرها) تا در نهایت به واحدهای کنترل کننده حافظه تحویل شوند. در سمت کنترل کننده حافظه، درخواست‌ها در صف پاسخ منتظر می‌مانند تا زمانی که حافظه به بسته‌های قدیمی پاسخ دهد. در اولین مرحله، حافظه نهان L2 برای فراهم کردن داده مناسب مورد بررسی قرار می‌گیرد، درصورت یافتن داده مورد نیاز، مستقیماً از طریق شبکه پاسخ به SM درخواست دهنده ارسال می‌شود در غیر این صورت، واحد کنترل کننده حافظه درخواست را در بافر حافظه اصلی قرار می‌دهد.

فرض شود که یک برنامه حافظه محور در حال اجرا می‌باشد و تعداد زیادی درخواست حافظه تولید کرده است که شبکه میان‌ارتباطی را نزدیک به حالت اشباع قرار داده است. در این شرایط نرخ نزاع در اطراف گره‌های کنترل کننده حافظه افزایش می‌یابد و درخواست‌ها زمان بیشتری را برای دریافت منابع و پورت مورد نظر منتظر می‌مانند. شبکه میان‌ارتباطی به عنوان یکی از مهمترین قسمت‌های مسیر دسترسی به حافظه نقش مهمی در عملکرد کلی GPU ایفا می‌کند. در برخی تحقیقات، نقش شبکه میان‌ارتباطی در میانگین تأخیر دسترسی حافظه تا ۵۰٪ گزارش شده است [۲۷]. در این مقاله ما بر کاهش تاثیرات منفی شبکه میان‌ارتباطی به نفع درخواست‌های تولید شده هسته‌های بحرانی که نزدیک به حالت بیکاری قرار دارند تمرکز می‌کنیم. بسته‌های بحرانی به درخواستهایی گفته می‌شود که توسط SM ای تولید می‌شود که تعداد warp اماده کمتری در صف آن وجود دارد. به عبارت دیگر، یک SM که تعداد بیشتری درخواستهای حافظه پاسخ داده نشده دارد، شرایط بحرانی تری خواهد داشت. بنابراین این SMها با احتمال بسیار به حالت بیکار فرو خواهند رفت.

به این ترتیب، ایده اصلی رویکرد پیشنهادی ما شناسایی SMهای بحرانی و اولویت دادن به درخواست‌های آن‌ها در شبکه میان‌ارتباطی و صف‌های کنترل کننده حافظه است. این روش تأخیر مسیر دسترسی به حافظه را برای درخواستهای بحرانی به حداقل می‌رساند تا از میزان بیکاری SMهای فعال جلوگیری نماید. با این روش،

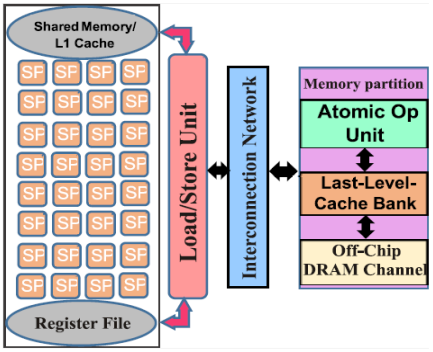
میانگین زمان بیکاری SMها کاهش یافته و عملکرد GPU به طور قابل توجهی بهبود می‌یابد. سطح اولویت اعطایی به بسته‌ها با توجه به بحرانی بودن SMهای تولید کننده آن‌ها تعیین می‌شود. به منظور تخصیص منابع و اختصاص کانال مجازی، در ابتدا بسته‌های بحرانی‌تر منابع مورد نظر خود را دریافت می‌کنند. با این حال، در این روش، تأخیر متوسط شبکه ممکن است تغییر نکند ، اما توزیع تأخیر به نفع بسته‌های دارای اولویت بالا تغییر خواهد کرد (به عنوان مثال، به طور متوسط درخواست‌های با اولویت بالاتر نسبت به درخواست با اولویت کمتر، تاخیر دسترسی به حافظه کمتری دارند) تا تعادل بین SMها از نظر میزان بحرانی بودن حفظ شود.

۴- طرح پیشنهادی

در این مقاله سعی شده است تا تأثیر کاهش میزان بیکاری SMها در عملکرد پردازنده‌های گرافیکی نشان داده شود. در GPUها، اطلاعات مربوط به ماهیت و وضعیت درخواست‌های حافظه در جداول وضعیت ذخیره می‌شود، بنابراین واحدهای برنامه ریز درخواست‌ها را بر اساس این جداول مدیریت می‌کنند. در این مطالعه، ما از چنین اطلاعات ذخیره شده در این جداول برای شناسایی SMهای بحرانی که نزدیک به حالت بیکاری هستند استفاده کرده ایم. در واقع، در این پژوهش به درخواست SMهای بحرانی در بخش‌های مختلف شبکه میان ارتباطی برای دریافت منابع درخواستی اولویت بالاتری تخصیص داده می‌شود. به منظور ارزیابی اثرات رویکرد پیشنهادی، معیارهایی مانند تعداد کل چرخه‌های زمانی اجرا و تعداد کل دستورالعمل‌ها در هر چرخه گزارش شده است.

۴-۱- روش تحقیق

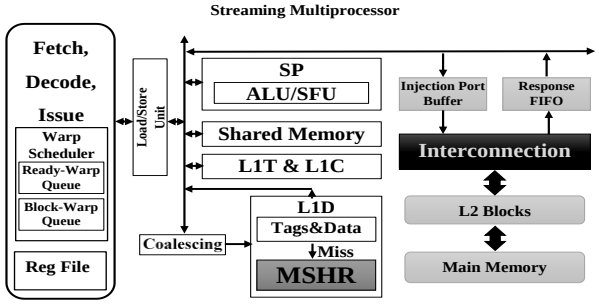
منابع در پردازنده‌ها به مولفه‌هایی گفته می‌شود که چندین درخواست برای به دست آوردن آنها در یک لحظه خاص وجود دارد. به عبارت دیگر، بین درخواست‌ها برای به دست آوردن این مولفه‌ها رقابت وجود دارد و اگر یکی از متقاضیان بازنده رقابت باشد، باید در صف انتظار بماند تا زمانیکه زمان بند سرانجام نوبت را به آن اختصاص دهد. در یک دسته بندی کلی، در سیستم‌های چند پردازنده‌ای می‌توان منابع را به دو بخش پردازشی و ذخیره سازی طبقه بندی نمود.



شکل ۲: نمای کلی از معماری GPU پایه

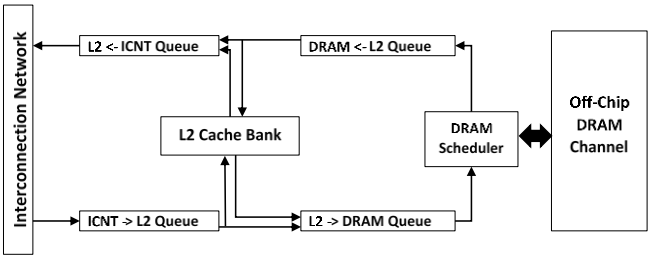
همانطور که قبلاً ذکر شد، پردازنده‌های گرافیکی مفهوم "Single Instructive Multiple Threads (SIMT) را اجرا می‌کنند و هر SM پردازنده‌های جریانی خود را با عرض SIMT دارد. شکل ۲ نمای کلی از معماری GPU همراه با سطوح مختلف ارتباطات را نشان می‌دهد. سلسله مراتب حافظه در GPUها زمانبندی خاص خود را دارند که با یکدیگر به طور مستقل کار می‌کنند. با ورود درخواست‌های حافظه از واحدهای پردازشی به واحد LOAD/STORE، درخواست‌ها در حافظه نهان L1 بررسی می‌شوند و اگر دسترسی به حافظه نهان L1 موفقیت آمیز باشد، پاسخ سریعاً توسط SM متقاضی دریافت می‌شود و واحدهای پردازشی کار خود را ادامه می‌دهند. در غیر این صورت درخواست حافظه در صف درخواست‌های ناموفق L1 قرار می‌گیرد. هنگامی که تعداد درخواست‌های ناموفق L1 بیشتر از سایز صف درخواست ناموفق L1 باشد، درخواست‌های بعدی توسط کنترل کننده حافظه پنهان پذیرفته نمی‌شوند، درنتیجه SM به حالت بیکار فرو می‌رود تا زمانی که پاسخ متناسب برای ان از سمت حافظه اماده گردد.

Miss Status Holding Register که به اختصار MSHR نامیده می‌شود یکی از موثرترین پارامترها برای نظارت بر وضعیت هر SM است [۲۹،۲۸]. برای هر دسترسی ناموفق به حافظه‌نهان سطح یک، یک ردیف خاص به جدول MSHR اضافه می‌شود که مشخصات درخواست را در خود نگه می‌دارد. شکل ۳ موقعیت MSHR را در بخش حافظه‌نهان سطح یکSM نشان می‌دهد.



شکل ۳: جایگاهMSHR در SM

شکل ۴ سلسله مراتب حافظه GPU را به تصویر می‌کشد، هنگامی که دسترسی به حافظه‌نهانL2 موفقیت آمیز باشد، پاسخ پس از مراحل آماده سازی در صف L2 ICNT قرار می‌گیرد تا برای SM ارسال شود. در غیر این صورت درخواست‌ها به حافظه اصلی بافر منتقل می‌شوند. پس از آماده سازی داده‌ها در قسمت حافظه، پاسخ‌های مناسب پس از نوشتن آن در حافظه نهان L2 برای SM متقاضی ارسال می‌شود.



شکل ۴: ساختار حافظه GPU

۴-۲- رصد وضعیت درخواست‌های حافظه

به طور کلی ساختار GPU پایه به سه قسمت مختلف طبقه بندی می‌شود: (I) واحد پردازش با حافظه نهان اختصاصی. (II) حافظه نهان L2 و حافظه اصلی. (III) شبکه‌های میان‌ارتباطی. برای قسمت اول و دوم از تقسیم بندی فوق، فعالیتهای مختلفی جهت حفظ عملکرد GPU صورت گرفته است. به عنوان مثال، مدیریت سایز حافظه نهان L1 [۳۰]، اعمال برنامه ریزی بهینه در صف منابع حافظه [۳۱]، زمان بند بلاک‌های برنامه برای تعادل بخشیدن به درخواست‌های حافظه در GPU، اعمال برنامه ریزی چند سطحی Warpها . اما قسمت سوم تقریباً یک موضوع نوظهور در زمینه عملکرد GPU است.

در این مطالعه، ما قسمت اول و سوم را با موفقیت ترکیب کردیم تا کارایی GPU را با حداقل میزان سربار اضافی از لحاظ مساحت و توان مصرفی افزایش دهیم. در هر لحظه از اجرای برنامه، SMهای مختلف ممکن است درخواست‌های ناموفق به L1 داشته باشند که هنوز توسط واحد حافظه پاسخ داده نشده است. از نظر احتمال، SMهایی که تعداد درخواست‌های پاسخ داده نشده بیشتری دارد به احتمال بالاتری به حالت بیکار می‌روند. به عبارت دیگر، SMهایی که درخواست L1 از دست رفته‌ی

بیشتری دارند، وضعیت بحرانی‌تری نسبت به SMهای دیگر دارند و احتمال ورود آنها به حالت بیکاری بیشتر است. هنگامی که یک درخواست حافظه نهان L1 صادر می‌شود، اگر آدرس مورد نظر در حافظه نهان L1 وجود داشته باشد، برنامه ریز حافظه پنهان بلافاصله به متقاضی SM پاسخ می‌دهد در غیر این صورت، درخواست توسط حافظه پنهان سطح پایین تر یا حافظه اصلی پاسخ داده می‌شود. در دسترسی‌های موفق، تقریباً اخلالی در اجرای معمول دستورالعمل‌ها ایجاد نمی‌شود. اما به ازای دسترسی‌های ناموفق به حافظه نهان L1 مراحل پیچیده‌ای باید طی شود تا پاسخ اماده گردد. پس از بروز دسترسی ناموفق این درخواست در صف درخواست‌های ناموفق قرار می‌گیرد و جدول MSHR به روز می‌شود [۳۲-۳۴]. لازم به ذکر است که هر SM صف L1 و جدول MSHR مخصوص به خود را برای درخواستهای از دست رفته دارد. قبل از ورود یک درخواست جدید به صف درخواست‌های ناموفق L1، کنترل کننده حافظه نهان بلوک آدرس درخواست را با مابقی درخواست‌های موجود مقایسه می‌کند. اگر آدرس مشابهی پیدا نکند، درخواست جدید را به صف اضافه میکند، در غیر این صورت درخواست جدید را با درخواست مشابه موجود ادغام میکند [۳۵،۳۶].

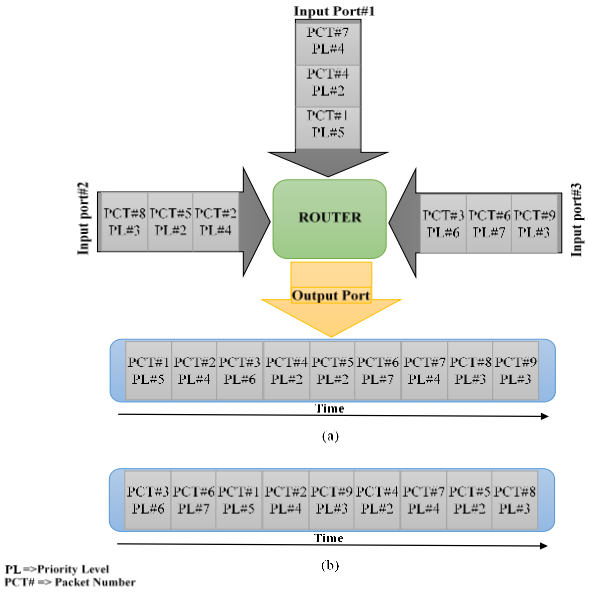
اگر جدول MSHR مربوط به یک SM به دلیل تعداد بالای دسترسی ناموفق حافظه نهان L1 پر شود، SM به حالت بیکار وارد می‌شود و تا زمان دریافت داده مناسب از سطوح حافظه پایین‌تر در این حالت باقی می‌ماند. وقتی تعداد دسترسی‌های ناموفق به حافظه نهان سطح L1 از سایز صف درخواست از دست رفته بیشتر باشد، مشخصات درخواست‌های نادیده گرفته شده در شمارنده Reservation Failed ذخیره می‌شود. مقادیر ثبت شده در این شمارنده می‌تواند به عنوان منبعی برای تجزیه و تحلیل ماهیت برنامه‌ها و وضعیت SMها استفاده شود. معمولا در الگوریتم‌های مسیریابی شبکه میان ارتباطی همه بسته‌ها دارای اولویت یکسانی هستند و هیچ یک بر دیگری برتری ندارد. این یک روش معمول در بیشتر پردازنده‌های گرافیکی است. همه بسته‌ها در داوری‌ها و تخصیص منابع در شبکه از هویت یکسانی برخوردار هستند.

در این پژوهش، میزان بحرانی بودن هر SM با نظارت بر اطلاعات موجود در رجیسترهای کنترلی مانند MSHR شناسایی می‌شود. در هرگام از فرایند مسیریابی، درخواست SMهای بحرانی‌تر اولویت بیشتری برای به دست آوردن منابع در شبکه میان ارتباطی خواهند داشت. به این ترتیب، درخواست‌های با اولویت بالاتر سریعتر از بسته‌های دیگر به گره‌های کنترل کننده حافظه تحویل داده می‌شوند. در واقع با این رویکرد، پاسخ به درخواست‌های بحرانی سریعتر از حالت معمول انجام می‌شود درنتیجه، SMهایی که شرایط بحرانی دارند مقاومت بیشتری برای ورود به حالت بیکاری خواهند داشت. به عبارت دیگر، SMها در برابر تاخیر دسترسی به حافظه، قابلیت تحمل پذیری بالاتری دارند.

۳-۴- داوری آگاه به Miss-Status (MSAA)

(Miss Status Aware Arbitration)

در یک دسته بندی کلی روش پیشنهادی ما در دو مرحله اساسی ارائه شده است. (I) در معماری پایه سایز صف درخواست‌های ناموفق حافظه‌نهان سطح یک و جدول MSHR 32 است. همچنین ۸ سطح اولویت (۰ تا ۷) برای بسته‌هایی که به شبکه ارتباطی تزریق می‌شوند درنظر گرفته شده است. در واقع سطح اولویت از تقسیم تعداد سطرهای پرشده جدول MSHR به عدد ۴ بدست خواهد امد. به عنوان مثال، اگر یک SM خاص دارای شمارنده 9 برای MSHR باشد و یک SM دیگر دارای شمارنده ۱۰، در این ارزیابی هر دو SM ذکر شده دارای اولویتی برابر۳ هستند. مقادیر بالاتر شمارنده MSHR به این معنی است که SM مربوطه از لحاظ نزدیکی به حالت بیکاری در وضعیت بحرانی‌تری قرار دارد. سطح اولویت ۷ به معنای بالاترین اولویت است که توسط یک SM بحرانی تولید شده است بنابراین این بسته برای تخصیص منابع بالاترین اولویت را دارد. (II) برای دستیابی به بهترین نتیجه در این تحقیق، سطوح اولویتی که برای تسریع درخواست SMهای بحرانی به بسته ان‌ها اضافه می‌شود، دقیقاً مشابه با سایر وضعیت‌های کنترلی در بسته‌های پاسخ نیز قرار می‌گیرد. با استفاده از این رویکرد، اهمیت درخواستهای دارای اولویت بالا در هر دو شبکه درخواست و پاسخ حفظ می‌شود. شکل ۵ وضعیت عادی اطراف روترها را در شبکه میان ارتباطیGPU نشان می‌دهد. در شکل ۵.(a) روال عادی تصمیم گیری تخصیص پورت نمایش داده شده است. پورت ورودی ۱، پورت ورودی ۲ و پورت ورودی ۳ برای بدست آوردن پورت خروجی در یک چرخه خاص رقابت میکنند. در چرخه اول بسته شماره ۱ از پورت ورودی ۱، بسته شماره ۲ از پورت ورودی ۲ و بسته شماره ۳ از پورت ورودی ۳ درگاه خروجی را درخواست کرده‌اند، داور پورت خروجی را با توجه به شماره بسته (زمان ورود) اختصاص می‌دهد، ابتدا بسته شماره ۱ از پورت خروجی برخوردار می‌شود در مرحله دوم بسته شماره ۲ پورت خروجی را دریافت می‌کند و سپس بسته شماره ۳ از پورت خروجی عبور می‌کند. به همین ترتیب، در چرخه‌های دوم و سوم تخصیص پورت خروجی به ترتیب شماره بسته‌ها صورت می‌گیرد. درنهایت با توجه به آرایش بسته‌ها در درگاه ورودی مسیریابی معمولی، پورت خروجی را به همان ترتیب که شکل ۵.(a) نشان داده است انجام می‌دهد. با توجه به شکل بسته شماره ۱ قبل از همه از درگاه خروجی عبور کرده و سپس بسته شماره ۲ و در اخر بسته شماره ۹ موفق به دریافت درگاه خروجی شده است. در شکل ۵.(b) نتیجه اعمال روش MSAA نشان داده شده است. در چرخه اول، هر یک از ۳ پورت ورودی برای گرفتن پورت خروجی درخواست داده‌اند، علیرغم تصمیم گیری مسیریابی متداول، پورت ورودی ۳ که دارای سطح اولویت # ۶ است، بدون در نظر گرفتن ترتیب درخواست، پورت خروجی را دریافت می‌کند.



شکل ۵: سیاست مسیریابی شبکه میان‌رتباطی GPU. (a) رفتار متعارف. (b) رویکرد MSAP.

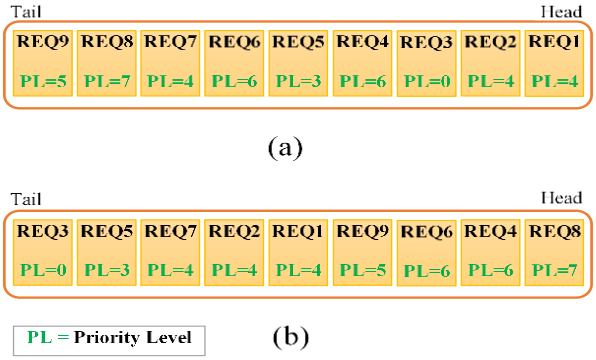
با توجه به رفتار متعارف در شکل ۵.(a)، بسته شماره ۳ باید از پورت خروجی در چرخه سوم عبور می‌کرد اما به منظور اولویت‌دهی به بسته‌های بحرانی با توجه به مدل MSAA در چرخه اول پورت خروجی را بدست اورده است. به همین ترتیب، در مسیریابی معمولی بسته شماره ۶ در چرخه ششم مسیریابی شده است اما در طرح MSAA این بسته در چرخه دوم مسیریابی می‌شود. همانطور که در شکل ۵.(b) مشاهده می‌شود، در میان بسته‌هایی که یک پورت مشترک را درخواست کرده‌اند، MSAA پورت را به بسته‌ای می‌رساند که دارای بالاترین سطح اولویت است.

۴-۴- مرتب سازیِ آگاه به Miss-Status (MSAR)

(Miss Status Aware Reordering)

اگر بسته‌های درخواستی که در بافرهای مسیر حافظه منتظر هستند به گونه‌ای جایگیری مجدد شوند که درخواستی که توسط SMهای بحرانی‌تر تولید می‌شود، مقدم بر درخواست با SMهای کمتر بحرانی قرار گیرد، میانگین زمان توقف SMها به طور چشمگیری کاهش می‌یابد. به منظور ارتقاع عملکرد پردازنده گرافیکی در این پژوهش، درخواست‌ها براساس سطح اولویت اعطا شده ای که در بخش قبلی شرح داده شده، مرتب می‌شوند.

در واقع هنگامی که بسته‌های درخواست از شبکه خارج می‌شوند، براساس سطح اولویت در بافر حافظه دوباره مرتب می‌شوند. هنگامی که درخواستی از شبکه خارج می‌شود قبل از قرارگیری در بافر حافظه، کنترل کننده حافظه سطح اولویت درخواست را بررسی می‌کند تا مکان مناسب با سطح بحرانی بودن آن را پیدا کند. بافرهای حافظه معمولی به صورت FIFO کار می‌کنند، به این معنی که بسته‌های ورودی که زودتر وارد بافر شده‌اند زودتر هم بافر را ترک می‌کنند. در طرح پیشنهادی ما، بسته‌ها با توجه به ترتیب ورودشان بافر را ترک نمیکنند بلکه بسته ای که دارای سطح بالاتری از اولویت است، بدون توجه به ترتیب ورود، زودتر بافر را ترک می‌کند. ممکن است درخواستی که به عنوان آخرین بسته به بافر وارد شده قبل از همه، بافر را ترک کند زیرا توسط یک SM بحرانی تولید شده است که دارای بالاترین سطح اولویت در میان درخواست‌های موجود است.



شکل ۶: ترتیب قرار گرفتن بسته‌ها در صف حافظه پس از خارج شدن ازشبکه میان ارتباطی. (a) جایگیری اصلی.(b) جایگیری بر اساس MSAP.

شکل ۶ روش جایگیری مجدد پیشنهادی ما را نشان می‌دهد. در شکل ۶.(a) یک نمونه قرارگیری معمولی در بافر حافظه نشان داده شده است. بسته‌های خارج شده از شبکه بدون در نظر گرفتن سطح اولویت و فقط طبق منطق FIFO در صف قرار می‌گیرند و بافر حافظه را ترک می‌کنند. در این شکل فرض شده است، REQ9 آخرین بسته ای است که در بافر حافظه قرار می‌گیرد و REQ1 اولین درخواستی است که وارد بافر می‌شود. در ابتدا REQ1 و سپس REQ2 و در آخر REQ9 به ترتیب، بافر حافظه را ترک می‌کنند. شکل ۶.(b) جایگیری بسته‌های با اعمال تکنیک MSAR را نشان می‌دهد. یک بسته جدید هنگامی که به بافر حافظه وارد می‌شود، سطح اولویت آن با بسته‌های موجود مقایسه می‌شود و بعد از بسته‌های با سطح اولویت بالاتر و قبل از بسته‌های با سطح اولویت پایین تر قرار می‌گیرد. همانطور که از شکل مشخص است، در جایگذاری معمولی REQ1 بافر حافظه را در اولین چرخه خروجی ترک می‌کند اما در طرح MSAR، REQ1 حافظه را در سیکل خروجی پنجم ترک میکند. REQ8 و REQ9 به ترتیب دارای سطح اولویت ۷ و ۵ هستند، در بافر معمولی این بسته‌ها در دوره ۸ و ۹ از بافر خارج می‌شوند اما در طرح MSAR این بسته‌ها به عنوان بسته‌های اول و چهارم مرتب شده‌اند و بافر حافظه را در چرخه اول و چهارم ترک می‌کنند.

۵- مسیر پژوهش

طرح پیشنهادی ما با شبیه ساز GPGPU-Sim 3.2.0 [۳۷] که یک شبیه ساز cycle-accurate است ارزیابی شده است. در این پژوهش از ۱۴ برنامه گوناگون از دو مجموعه برنامه محک ISPASS [۳۷] و Rodinia [۳۸] استفاده شده است. برنامه محک‌ها به گونه‌ای انتخاب شده‌اند تا تمامی ابعاد طرح پیشنهادی را مورد بررسی قرار دهند. از لحاظ معماری، شبیه‌ساز تقریبا مشابه به معماری NVIDIA's FermiGTX 480 تنظیم شده است. همانگونه که از جدول ۱ مشخص است، تنظیمات پایه شامل ۱۶KB 4-way set-associative حافظه نهان سطح اول برای هر SM است و یک ۸x128KB 8-way set-associative حافظه نهان سطح دوم است. توپولوژِی شبکه میان ارتباطی یک Mesh دو بعدی ۸*۸ است که شامل ۸ گره کنترل کننده حافظه و ۵۶ نود SM است.

برنامه محک‌های Rodinia به صورت عمده از حافظه مشترک استفاده می‌کنند و حساس به حافظه نهان نیستند. طرح پیشنهادی ما به ازای برنامه‌های حساس به حافظه نهان بهبود بیشتری را حاصل می‌کند. برنامه محک‌های ISPASS بسیار حساس به حافظه اصلی و حافظه نهان هستند. بنابراین طرح پیشنهادی ما به ازای این برنامه محک‌ها بهبود چشمگیرتری خواهد داشت.

جدول ۱: پارامترهای تنظیم شده در شبیه ساز GPGPU-Sim

Parameter	Value
Number of SM	٥٦
Warp size	٣٢ Threads
Per-SM	١٠٢٤, ٤٨ Warp, ٨ Thread Blocks, ٣٢ MSHR
Number of LLC bank	٨
Number of MC	٨
Number of Registers / SM	٣٢٧٦٨ Bytes
Size of shared memory	٤٩١٥٢ Bytes
Size of L1 data cache / SM	١٦KB (٣٢ sets/٤ ways LRU)
Size of L2 cache	٨ x ١٢٨, ٨-way
Memory Clock	٩٢٤ MHz
Compute Core Clock	٧٠٠ MHz
Routing topology	٨ x ٨ ٢D Mesh
Routing	٤
Channel Latency	١
Virtual Channels	١

۶- نتایج

به دلیل ماهیت ساختار دسترسی به حافظه، شبکه‌های میان ارتباطی به عنوان یکی از گلوگاه‌های اصلی عملکرد پردازنده‌های گرافیکی شناخته می‌شوند. عدم تعادل ترافیک در شبکه‌های درخواست و پاسخ، نرخ نزاع را در میان درخواست‌های موجود در شبکه افزایش می‌دهد. از آنجا که تعداد واحدهای کنترل کننده حافظه بسیار کمتر از واحدهای SM است، ترافیک نامتعادل در شبکه میان ارتباطی پردازنده‌های گرافیکی به وجود می‌آید. با توجه به ساختار پردازنده‌های گرافیکی، SMها که تعداد ان‌ها بسیار بیشتر از واحدهای کنترل کننده حافظه است درخواست‌های خود را به ان‌ها می‌فرستند، از این رو اطراف گره‌های کنترل کننده حافظه الگوی ترافیکی سنگینی تشکیل می‌شود در نتیجه، درخواست‌ها برای به دست آوردن منابع مناسب باید مدت زمان طولانی را منتظر منابع مورد نیاز خود در صف‌های انتظار سپری کنند. در این تحقیق، ما سعی کردیم تاخیر مسیر دسترسی به حافظه را برای هسته‌های پردازشی بحرانی، با اعمال اولویت به درخواست‌های ان‌ها در شبکه میان ارتباطی کاهش دهیم.

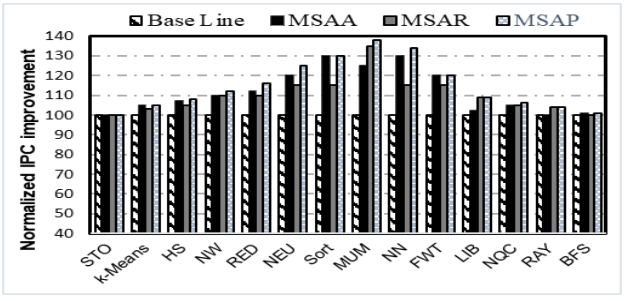
۶-۱- ارزیابی عملکرد

اعمال اولویت برای درخواست‌های حافظه با آگاهی از میزان دسترسی ناموفق به حافظه‌نهان سطح یک برای هر SM ایده اصلی این تحقیق است که ما ان را به اختصار Miss Status Aware Priority (MSAP)نام‌گذاری نمودیم. در این پژوهش به منظور دستیابی به نتایج قابل اطمینان، از برنامه محک‌های با رفتار مختلف در مدل اجرایی استفاده شده است. برنامه‌های حافظه محور، پردازش محور و خنثی از جمله انها است. در شکل ۷، IPC حاصل از اجرای برنامه محک‌های مختلف به ازای معماری پایه و MSAP مقایسه شده است. این شکل چهار نوار جداگانه را برای شرح دقیق تر روشهای پیشنهادی نشان می‌دهد. در واقع سه نوار دیگر به نوار پایه نرمال می‌شوند. نوار MSAA مقادیر IPC معیارهای مختلف را نشان می‌دهد هنگامی که فقط طرح MSAA اجرا شود بدون تکنیک MSAR. به همین ترتیب نوار MSAR، مقدار IPC را بدون اجرای روش MSAA نشان میدهد. نوار MSAP، مقادیر IPC را هنگام اجرای همزمان تکنیک MSAA و MSAR نشان می‌دهد. همان‌گونه که از شکل مشخص است MSAP موفق شده است که به‌ازای اکثر برنامه محک‌های حافظه محور بهبود چشمگیری ایجاد نماید. MUM به عنوان یکی از معیارهای حافظه محور دارای بیشترین بهبود IPC در بین معیارهای ارزیابی شده، نزدیک به ۳۸٪ در مقایسه با معماری پایه است.

علی‌رغم ماهیت حافظه محور برنامه محک BFS، روش MSAP نمی‌تواند زمان اجرای این معیار را بهبود بخشد. هنگامی که BFS روی GPU در حال اجرا است، تقریباً تمام SMها تعداد زیادی درخواست حافظه تولید می‌کنند و MSAP به اکثر SMها اولویت سطح بالایی می‌دهد. در این شرایط چون SMها تقریباً از اولویت تقریباً برابر برخوردار هستند، MSAP در افزایش سرعت چنین برنامه‌هایی کارایی ندارد. برنامه‌های پردازش محور، درخواست حافظه قابل توجهی ندارند و بیشتر حافظه مورد نیاز توسط حافظه حافظه‌نهان سطح یک برآورده می‌شود. STO و RAY به عنوان معیارهای پردازش محور دسته بندی می‌شوند. تقاضای حافظه آنها تقریباً کم است همواره تعداد کافی warp اماده برای اجرا وجود دارد. در این برنامه‌ها تاخیر حافظه گلوگاه عملکرد نخواهد بود بنابراین MSAP نقش مهمی در روند اجرای آنها ندارد. MSAP همچنین عملکرد GPU را برای معیارهای بی‌طرف مانند NQU و NW بهبود بخشیده است اما نه به اندازه برنامه‌های حافظه محور.

۶-۲- ارزیابی میزان Stall-Time

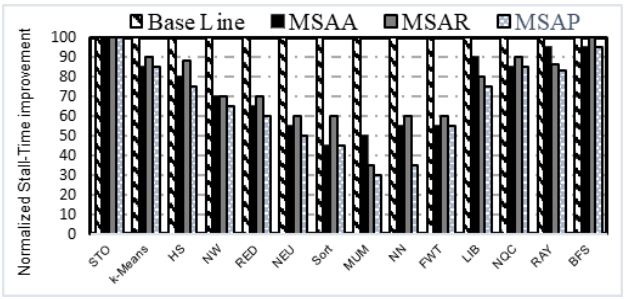
شکل ۸ میزان کلی چرخه بیکاری هسته‌ها را در طول اجرای برنامه محک‌ها نشان می‌دهد که به معماری پایه نرمال شده است. در این شکل برنامه محک MUM، که به بهبود قابل توجهی در IPC دست یافته است دارای بیشترین میزان بهبود در بیکاری SM نیز هست. هنگامی که MSAA بدون تکنیک MSAR اجرا می‌شود، میزان بیکاری SMها تقریباً ۵۰٪ کاهش می‌یابد و هنگامی که روش MSAR همراه با تکنیک MSAA اجرا می‌شود این عامل نزدیک به ۷۰٪ کاهش می‌یابد. همانطور که از شکل ۸ مشخص است، برای معیارهای محاسبه محور مانند STO پیشرفت چشمگیری در میزان بیکاری SMها وجود ندارد. به دلایل مشابه با بهبود IPC، بهبود زمان بیکاری SMها بیشترین تأثیر را بر برنامه محک‌های حافظه محور دارد.



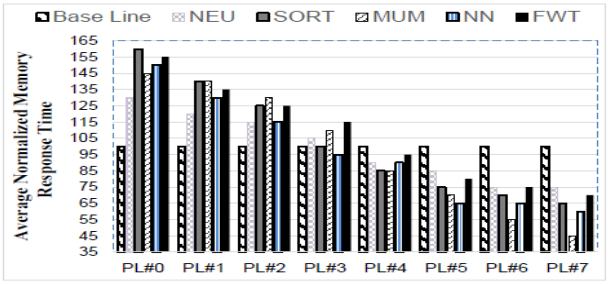
شکل ۷: مقایسه IPC معماری پایه با MSAP

۶-۳- متوسط زمان پاسخ گویی حافظه

در یک دسترسی عادی همه درخواست‌ها از اهمیت یکسانی برخوردار هستند و هیچ یک از آنها اولویتی بر دیگری ندارند. اما میانگین زمان پاسخ‌گویی حافظه در رویکرد MSAP برای همه درخواست‌های حافظه برابر نیست. درخواست‌هایی که از SMهای بحرانی تولید می‌شوند و دارای سطح اولویت بالاتر هستند، نسبت به بسته‌های سطح اولویت پایین، تاخیر حافظه کمتری خواهند داشت. شکل ۹ متوسط زمان پاسخگویی حافظه برای سطوح مختلف اولویت به ازای برنامه محک‌های مختلف را نشان می‌دهد. در این شکل تکنیک MSAP با معماری پایه از لحاظ میانگین تاخیر حافظه به ازای بسته‌ها با سطوح اولویت مختلف مقایسه شده است. همانطور که از شکل ۹ مشخص است، متوسط زمان پاسخگویی حافظه برای بسته‌های دارای اولویت پایین تر افزایش می‌یابد و از طرف دیگر به ازای بسته‌های دارای سطح اولویت بالاتر کاهش می‌یابد. در واقع مفهوم طرح پیشنهادی ما به وضوح در این شکل نمایان است، درخواست SMهای بحرانی باید سریعتر از درخواست SMهای غیر بحرانی پاسخ داده شود. به ازای برنامه محک MUM، میانگین زمان پاسخگویی حافظه برای بسته‌هایی با سطح اولویت صفر تقریباً ۴۵٪ افزایش یافته است و از طرف دیگر میانگین زمان پاسخگویی حافظه برای بسته‌هایی با سطح اولویت هفت، ۵۵٪ کاهش یافته است. مطابق شکل ۹، برنامه محک SORT دارای بدترین زمان پاسخگویی حافظه برای سطح اولویت صفر است زیرا تعداد زیادی از SMها در طی اجرای به وضعیت بحرانی می‌رسند، بنابراین در این معیار سطح اولویت پایین بیشترین آسیب پذیری را خواهد داشت.



شکل ۸: مقایسه میزان Stall-Time به ازای معماری پایه و MSAP



شکل ۹: متوسط زمان پاسخ گویی حافظه به بسته‌های با اولویت متفاوت

۷- کارهای مرتبط

کارهای مرتبط مختلفی وجود دارد تا با بهینه کردن مسیر دسترسی به حافظه اصلی تاخیر دسترسی و در نتیجه کارایی پردازنده گرافیکی را بهبود بخشد. در ادامه برخی از این پژوهش‌ها به صورت دسته بنده شده بررسی شده است.

مدیریت واحد کنترل کننده حافظه: Jog وهمکاران مدلی را ارایه کرده‌اند تا با بازنشانی درخواست‌ها در بافرهای حافظه به هسته‌های بحرانی اولویت بالاتری اعمال کنند [۳۹]. برخی از مطالعات تکنیک‌های زمان‌بندی حافظه را بر

اساس واگرایی درخواست‌ها پیش نهاد داده‌اند تا بتوانند با افزایش میزان محلیت بین درخواست‌ها کارایی پردازنده گرافیکی را بهبود دهند [۴۱،۴۰،۲۳،۲۲].

سیاست جایگذاری حافظه نهان: کارهای مختلف بر روی تکنیک جایگذاری حافظه نهان صورت گرفته است تا با کاهش تاثیر منفی واگرایی حافظه میزان دسترسی موفق به حافظه نهان را بهبود بخشد [۴۳،۴۲]. برخی از کارها سیاست جایگذاری مختلفی را برای CPUها پیشنهاد کرده‌اند [۴۵،۴۴] و برخی دیگر مدلی اگاه بر پیش واکشی و جایگذاری حافظه نهان معرفی کرده‌اند [۴۷،۴۶].

زمان‌بند warp: پژوهش‌های بسیاری تکنیک زمان‌بندی warp را به منظور کاهش زمان دسترسی به حافظه و افزایش کارایی پردازنده گرافیکی پیشنهاد کرده‌اند [۴۹،۴۸،۱۳]. در [۴۹] تکنیک CCWS از یک مدل تشخیص محلیت استفاده کرده است تا با شناسایی محلیت‌های از بین رفته در زمان‌بندهای دیگر ان‌ها را به کار بگیرد.

دور زدن حافظه نهان: در این روش مسیر مستقل برای برخی از درخواست‌ها در نظر گرفته می‌شود تا بدون قرار گرفتن در صف پاسخگویی حافظه نهان مستقیم به سطوح پایین‌تر برای پاسخگویی ارسال شود. در واقع در این روش کنترل کننده حافظه نهان، درخواست‌های پیش بینی شده را برای جلوگیری از سربار دسترسی ناموفق مستقیما به سطوح پایین تر ارسال می‌کند [۵۱،۵۰]. Jia و همکارانش روشی را برای دور زدن حافظه نهان سطح یک پیشنهاد کرده‌اند تا با پیش‌بینی درخواست‌های واگرا قبل از دستیابی به حافظه نهان برای ان‌ها مسیر مستقیم به سطوح پایین‌تر ایجاد شود [۵۲].

۸- نتیجه‌گیری

پردازنده‌های گرافیکی به قدرت پردازش موازی خود معروف هستند، اما وقتی warpها به دلیل تاخیر بالای حافظه مسدود می‌شوند، این قدرت اجرای موازی به طور چشمگیری کاهش می‌یابد. تأخیر دسترسی به حافظه به عنوان مهمترین چالش دستیابی به حداکثر توان پردازشی GPU، در برخی ازموارد بدون تغییر در تکنولوژی ساخت قابلیت بهبود یافتن را دارد. رویکردهای پیشنهادی ما در MSAP با نظارت و اعطای اولویت به SMهایی که نزدیک به حالت بیکاری هستند، اقدام به کاهش اثرات مخرب تاخیر دسترسی به حافظه بر روی کارایی مجموعه می‌کند. در بسیاری از تحقیقات، روشهای مختلفی برای حفظ فعالیت SMها جهت دستیابی به حداکثر توان پردازشیGPU ارائه شده است. علیرغم سهم بالای شبکه میان ارتباطی در تاخیر حافظه در GPU، تحقیقات کمی در این زمینه انجام شده است و مشکلات حل نشده‌ی بسیاری در این زمینه وجود دارد.

بر اساس ارزیابی‌های صورت گرفته، MSAP برای معماری‌هایی که ناحیه مشترک مانند حافظه پنهان سطح آخر یا حافظه اصلی را برای چندین هسته پردازشی مستقل فراهم می‌کنند کارامد خواهد بود. در سیستم‌های کارایی‌گرا که تعداد زیادی نخ پردازشی به صورت مستقل دستورالعمل‌ها را در قالب تکنیک SIMT اجرا می‌کنند با استفاده از تکنیک MSAP می‌توانند به بهبود چشمگیری دست پیدا کنند. علاوه بر این، علیرغم مسیر حافظه بسیار بهینه در آخرین نسل‌های GPU مانند Volta [۵۳]، MSAP می‌تواند چالش‌های بیکاری هسته‌های پردازشی را در این معماری‌ها نیز بهبود بخشد.

به عنوان پژوهش آتی، به منظور بهبود زمان پاسخگویی و کاهش تاخیر دسترسی به حافظه برای هسته‌های بحرانی، در هنگام بروز دسترسی ناموفق به حافظه نهان سطوح مختلف، قبل از خارج کردن بلوک مد نظر از حافظه نهان، درخواست‌های منتظر در صف پاسخگویی حافظه نهان بررسی گردند تا به ان‌هایی که به این بلوک برای داده مورد نظرشان نیاز دارند پاسخ داده شود و سپس مکانیزم جایگذاری کش به وظیفه خود عمل نماید. همچنین از این طرح می‌توان در حافظه اصلی هنگام بروز تعویض سطر نیز بهره برد. با اجرای این طرح نرخ دسترسی ناموفق به حافظه نهان به شدت کاهش می‌یابد و کارایی سیستم افزایش خواهد داشت.

۹- مراجع

[1] J. Macri, "AMD's next generation GPU and high bandwidth memory architecture: FURY," *in Hot Chips 27 Symposium (HCS)*, 2015 IEEE, 2015, pp. 1–26.

[2] M. M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "A Compiler Framework for Optimization of Affine Loop Nests for Gpgpus," *in Proceedings of the 22Nd Annual International Conference on Supercomputing*, 2008, pp. 225–234.

[3] M. M. Baskaran, J. Ramanujam, and P. Sadayappan, "Automatic C-to-CUDA code generation for affine programs," *in International Conference on Compiler Construction*, 2010, pp. 244–263.

[4] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, no. 5, pp. 7–17, 2011.

[5] D. B. KirkW and W. Hwu, "Programming massively parallel processors." *Morgan Kaufmann*, Burlington, MA, 2010.

[6] N. Corporation, "NVIDIA's next generation CUDA compute architecture: Fermi," 2009.

[7] A. Munshi, "The opencl specification," *in Hot Chips 21 Symposium (HCS)*, 2009 IEEE, 2009, pp. 1–314.

[8] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[9] M. Bauer, H. Cook, and B. Khailany, "CudaDMA: optimizing GPU memory bandwidth via warp specialization," *in Proceedings of 2011 international conference for high performance computing, networking*, storage and analysis, 2011, p. 12.

[10] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, 2008.

[11] M. Lee, S. Song, J. Moon, and J. Kim, "Improving GPGPU resource utilization through alternative thread block scheduling," *in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 260–271.

[12] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-effective on-chip networks for manycore accelerators," *in Proceedings of the 2010 43rd annual IEEE/ACM international symposium on microarchitecture*, 2010, pp. 421–432.

[13] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," *in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 308–317.

[14] A. Sethia, D. A. Jamshidi, and S. Mahlke, "Mascar: Speeding up GPU warps by reducing memory pitstops," *in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 174–185.

[15] M. Abdel-Majeed and M. Annavaram, "Warped register file: A power efficient register file for GPGPUs," *in 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 412–423.

[16] Y. Zhang, Z. Xing, C. Liu, C. Tang, and Q. Wang, "Locality based warp scheduling in GPGPUs," *Futur. Gener. Comput. Syst.*, vol. 82, pp. 520–527, 2018.

[17] X. Cheng, H. Zhao, S. P. Mohanty, and J. Fang, "Improving GPU NoC Power Efficiency through Dynamic Bandwidth Allocation," *in 2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–4.

[18] A. Jog, O. Kayiran, N. Chidambaram Nachiappan, and A. K. Mishra, "OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance," *in ACM SIGPLAN Notices, 2013*, vol. 48, no. 4, pp. 395–406..

[19] Y. Yu, X. He, H. Guo, Y. Wang, and X. Chen, "A credit-based load-balance-aware CTA scheduling optimization scheme in GPGPU," Int. J. *Parallel Program.*, vol. 44, no. 1, pp. 109–129, 2016.

[20] A. Bakhoda, J. Kim, and T. M. Aamodt, "Designing on-chip networks for throughput accelerators," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 3, p. 21, 2013.

[21] J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 4, p. 48, 2013.

predictions," *in 2003 International Conference on Parallel Processing, 2003. Proceedings.*, 2003, pp. 294–302.
[44] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutluy, and D. A. Jimenezz, "Improving cache performance using read-write partitioning," *in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 452–463.
[45] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 381–391, 2007.
[46] V. Seshadri, H. Xin, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "Mitigating prefetcher-caused pollution using informed caching policies for prefetched blocks," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 1–22, 2015.
[47] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared resource management for multi-core systems," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 141–152, 2011.
[48] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Divergence-aware warp scheduling," *in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 99–110.
[49] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," *in Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 72–83.
[50] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney, and J. Nuzman, "Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches," *in Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012, pp. 293–304.
[51] J. Gaur, M. Chaudhuri, and S. Subramoney, "Bypass and insertion algorithms for exclusive last-level caches," *in Proceedings of the 38th annual international symposium on Computer architecture, 2011*, pp. 81–92.
[52] W. Jia, K. A. Shaw, and M. Martonosi, "MRPB: Memory request prioritization for massively parallel processors," *in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 272–283.
[53] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA volta GPU architecture via microbenchmarking," *arXiv Prepr. arXiv1804.06826*, 2018.

حسین بی‌طالبی مدرک دوره کارشناسی را در رشته مهندسی کامپیوتر از دانشگاه اصفهان در شهریور ماه ۱۳۹۳ دریافت نموده است. وی مدرک کارشناسی ارشد خود را از دانشگاه تهران در شهریور ماه ۱۳۹۶ اخذ کرده است. در حال حاضر او دانشجوی مقطع دکتری در دانشگاه شهید بهشتی است. برخی از علایق پژوهشی او به شرح زیر می باشد: کارایی پردازنده های گرافیکی، معماری کامپیوتر، شتاب دهنده‌های سخت‌افزاری، محاسبات تخمینی و شبکه های میان ارتباطی می باشد. آدرس پست الکترونیکی ایشان عبارت است از: h_bitalebi@sbu.ac.ir

فرشاد صفایی کارشناسی، کارشناسی ارشد و دکتری خویش را در مهندسی کامپیوتر، معماری سیستم‌های کامپیوتری از دانشگاه علم و صنعت ایران، به ترتیب، در سال‌های ۱۳۷۳، ۱۳۷۶ و ۱۳۸۶ دریافت کرد. او اکنون با درجه دانشیاری در دانشکده مهندسی و علوم کامپیوتر دانشگاه شهید بهشتی به تدریس و پژوهش اشتغال دارد. زمینه‌های پژوهشی مورد علاقه او عبارتند از مدل‌سازی و ارزیابی کارآیی سیستم‌های کامپیوتری، شبکه‌های پیچیده و اجتماعی، شبکه‌های میان ارتباطی و سامانه‌های ذخیره-سازی پیشرفته. آدرس پست الکترونیکی ایشان عبارت است از: f_safaei@sbu.ac.ir

[22] R. Ausavarungnirun, S. Ghose, O. Kay, M. T. Kandemir, and O. Mutlu, "Holistic management of the GPGPU memory hierarchy to manage warp-level latency tolerance," *arXiv Prepr. arXiv1804.11038*, 2018.
[23] N. Chatterjee, M. O'Connor, G. H. Loh, N. Jayasena, and R. Balasubramonian, "Managing DRAM latency divergence in irregular GPGPU applications," *in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 128–139.
[24] N. B. Lakshminarayana and H. Kim, "Effect of instruction fetch and memory scheduling on gpu performance," *in Workshop on Language, Compiler, and Architecture Support for GPGPU*, 2010, vol. 88.
[25] N. Brunie, S. Collange, and G. Diamos, "Simultaneous branch and warp interweaving for sustained GPU performance," *in 2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 49–60.
[26] Y. Yu, W. Xiao, X. He, H. Guo, Y. Wang, and X. Chen, "A stall-aware warp scheduling for dynamically optimizing thread-level parallelism in GPGPUs," *in Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 15–24.
[27] P. Lotfi-Kamran, M. Modarressi, and H. Sarbazi-Azad, "Near-Ideal Networks-on-Chip for Servers," *in High Performance Computer Architecture (HPCA)*, 2017 IEEE International Symposium on, 2017, pp. 277–288.
[28] Y. Oh, K. Kim, Y. Park, W. W. Ro, and M. Annavaram, "APRES: Improving cache efficiency by exploiting load characteristics on GPUs," *ACM SIGARCH Comput. Archit. news*, vol. 44, no. 3, pp. 191–203, 2016.
[29] F. Candel, S. Petit, J. Sahuquillo, and J. Duato, "Accurately modeling the on-chip and off-chip GPU memory subsystem," *Futur. Gener. Comput. Syst.*, vol. 82, pp. 510–519, 2018.
[30] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access pattern-aware cache management for improving data utilization in gpu," *in ACM SIGARCH Computer Architecture News*, 2017, vol. 45, no. 2, pp. 307–319.
[31] L. Wang, J. Ye, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: dynamic GPU memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2018, pp. 41–53.
[32] X. Chen, L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W.-M. Hwu, "Adaptive cache management for energy-efficient gpu computing," *in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 343–355.
[33] G. B. Kim, J. M. Kim, and C. H. Kim, "MSHR-Aware Dynamic Warp Scheduler for High Performance GPUs," *KIPS Trans. Comput. Commun. Syst.*, vol. 8, no. 5, pp. 111–118, 2019.
[34] Y. Gu and L. Chen, "Dynamically linked MSHRs for adaptive miss handling in GPUs," *in Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 510–521.
[35] D. Kroft, "Cache memory organization utilizing miss information holding registers to prevent lockup from cache misses." *Google Patents*, 1983.
[36] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," *in 25 years of the international symposia on Computer architecture (selected papers)*, 1998, pp. 195–201.
[37] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," *in 2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.
[38] S. Che, M. Boyer, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," *in 2009 IEEE international symposium on workload characterization (IISWC)*, 2009, pp. 44–54.
[39] A. Jog, O. Kayiran, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting core criticality for enhanced GPU performance," *in ACM SIGMETRICS Performance Evaluation Review*, 2016, vol. 44, no. 1, pp. 351–363.
[40] S. Mu, Y. Deng, Y. Chen, W. Zhang, and Z. Wang, "Orchestrating cache management and memory scheduling for GPGPU applications," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 8, pp. 1803–1814, 2013.
[41] G. L. Yuan, A. Bakhoda, and T. M. Aamodt, "Complexity effective memory access scheduling for many-core accelerator architectures," *in 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 34–44.
[42] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 60–71, 2010.
[43] J. Jalminger and P. Stenstrom, "A novel approach to cache block reuse

The CSI Journal on Computing Science and Information Technology

The CSI Journal on
Computing Science and
Information Technology
Vol. 19, No.1, 2021

Improve average memory access time for memory-intensive applications to reduce GPU stall time

Hossein BiTalebi, Farshad Safaei

Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.

Abstract

The emergence of GPGPU concept, coupled with CUDA and OpenCl programming models, has opened up new possibilities for reducing latency and power consumption in throughput-oriented workloads. GPU can execute thousands of parallel threads to hide the costly memory access latency. However, for some memory-intensive workloads, it is very likely that in some time intervals, all threads of a core are stalled and waiting for their data to be provided by the memory system. In this research, we aim to improve the GPU memory access latency to increase the thread activity time that results in decreasing core underutilization. In order to improve non-optimal time of cores we focus on the interconnection network between the cores and last level cache and prioritize the packet of those cores that have more number of stalled threads. This way, the highest priority in arbitrations and resource allocation is granted to more critical packets, so the memory request/reply of them will be serviced faster to reduce the overall core stall time. With this scheme, we can increase the throughput of a GPUGPU without increasing the interconnection network bandwidth: instead, we change the interconnection network bandwidth allocation policy in favor of SMs with less tolerance to interconnection network latency in order to hide or shorten the inactive SM periods.

Keywords: Cache contention; Graphics processing unit (GPU); cores stall time; interconnection network; locality; arbitration; row switching; priority.