دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی
مهندسی نرم‌افزار

تحلیلی بر عملکرد حافظه مشترک تحت بارهای کاری چند منظوره در پردازنده‌های گرافیکی

نگارش
پارسوآ خورسند رحیم‌زاده

استاد راهنما
دکتر حمید سربازی آزاد

۴ تیر ۱۳۹۶

@title

چکیده

در پردازنده‌های گرافیکی روی هر چندپردازنده جریانی یک واحد حافظه اختصاصی برای نگهداری داده‌های مربوط به فضای آدرس‌دهی مشترک ریسمان‌های در حال اجرا تعبیه شده‌است. فضای این حافظه به بلوک‌های هر چندپردازنده به طور جداگانه تخصیص می‌یابد و میان ریسمان‌های هر بلوک به طور مشترک استفاده می‌شود.

حافظه مشترک به برنامه‌نویس اجازه می‌دهد تا اطلاعات مورد نیاز برای اجرای هر بلوک از ریسه‌های برنامه را روی چندپردازنده جریانی شامل آن‌ها ذخیره کند. حافظه مشترک به دلیل ظرفیت و در نتیجه مساحت کم و نیز قرار گرفتن روی تراشه چندپردازنده زمان دسترسی بسیار پایینی (در حد چند کلاک) دارد. به این ترتیب برنامه‌نویس می‌تواند بخشی از داده را که احتمال می‌دهد قرار است در باز زمانی فعلی با نرخ بالا مورد دسترسی قرار گیرد را روی این حافظه بارگذاری کند تا دسترسی به آن با سربار کم امکان‌پذیر باشد.

در نسل‌های اولیه پردازنده‌های گرافیکی، حافظه‌مشترک به عنوان راه‌حلی برای مدیریت پیچیدگی‌های ناشی از زمان دسترسی غیرقابل پیش‌بینی حافظه اصلی مورد پیاده‌سازی قرار گرفت. چنین رویکردی در کاربردهای گرافیکی که نیاز به تضمین نرخ فریم ثابتی وجود دارد از توجیه‌پذیری بالایی برخوردار است.

در ادامه نتایج تحلیل روی عملکرد و میزان کاربرد حافظه مشترک در بارهای کاری محاسباتی ارائه می‌شود و در نهایت راهکارهایی برای بهبود کارایی این مدل حافظه پیشنهاد می‌گردد .

کلمات کلیدی: پردازنده‌های گرافیکی عام‌منظوره، حافظه مشترک، کارایی، زمان دسترسی، فرکانس ، موازات سطح ریسمان .

فهرست مطالب

۱ مقدمه **۱**

۱.۱ پردازنده‌های چندهسته‌ای ۱

۲.۱ پردازنده‌های گرافیکی ۳

۳.۱ ساختار پایان‌نامه ۵

۲ مفاهیم پایه **۷**

۱.۲ دلایل روی آوردن به پردازنده‌های چندهسته‌ای ۷

۱.۱.۲ چگالی توان ۸

۲.۱.۲ دیوار حافظه ۸

۲.۲ پردازنده‌های چندهسته‌ای به عنوان یک راه‌حل ۹

۳.۲ پردازنده‌های گرافیکی عام‌منظوره ۹

۴.۲ استفاده از پردازنده گرافیکی برای مسائل عام‌منظوره ۹

۳ کارهای پیشین **۱۳**

۴ انگیزه و شهود **۱۵**

۱.۴ حافظه چرک‌نویس ۱۵

۲.۴ حافظه مشترک ۱۶

۳.۴ شبیه‌ساز GPGPUSim ۱۶

۱.۳.۴ ثبت دسترسی‌ها به حافظه در شبیه‌ساز ۱۶

۴.۴ بارهای کاری مورد استفاده ۱۸

۱.۴.۴ دسترسی به حافظه مشترک در بارهای کاری مورد بررسی و انگیز اولیه . ۱۸

۵.۴ معماری جایگزین پیشنهادی ۲۱

۵ نتایج بررسی **۲۳**

۱.۵ جمع آوری داده ۲۳
۲.۵ عملکرد حافظه نهان برای آدرس‌های حافظه اصلی پردازنده گرافیکی ۲۴
۳.۵ عملکرد حافظه نهان برای آدرس‌های حافظه مشترک ۲۹
۱.۳.۵ بررسی بهبود نرخ دستور در واحد زمان ۳۰

۶ جمع‌بندی و کارهای آینده **۳۳**

کتاب‌نامه **۳۵**

فهرست تصاویر

۱.۱ پیش‌بینی رشد چگالی توان پردازنده‌ها ۲
۲.۱ ساختار داخلی یک چندپردازنده جریانی در معماری Kepler ۴

۱.۲ تبعیت رشد تعداد ترانزیستورها از قانون مور ۱۰
۲.۲ شکاف بین عملکرد حافظه و پردازنده ۱۱
۳.۲ سلسله مراتب حافظه در یک پردازنده امروزی ۱۱
۴.۲ شکاف بین عملکرد پردازنده مرکزی و پردازنده گرافیکی در محاسبات عددی . . ۱۲

۱.۴ همبستگی نرخ IPC میان شبیه‌ساز و سخت‌افزار نسل Fermi ۱۷

۱.۵ روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری NN از پکیج ISPASS ۲۶
۲.۵ روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری SAD از پکیج پاربویل ۲۷
۳.۵ روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری Leukocyte از پکیج رودینیا ۲۸
۴.۵ نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک ISPASS ۳۰
۵.۵ نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک پاربویل ۳۱
۶.۵ نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک رودینیا ۳۱

۱ مقدمه

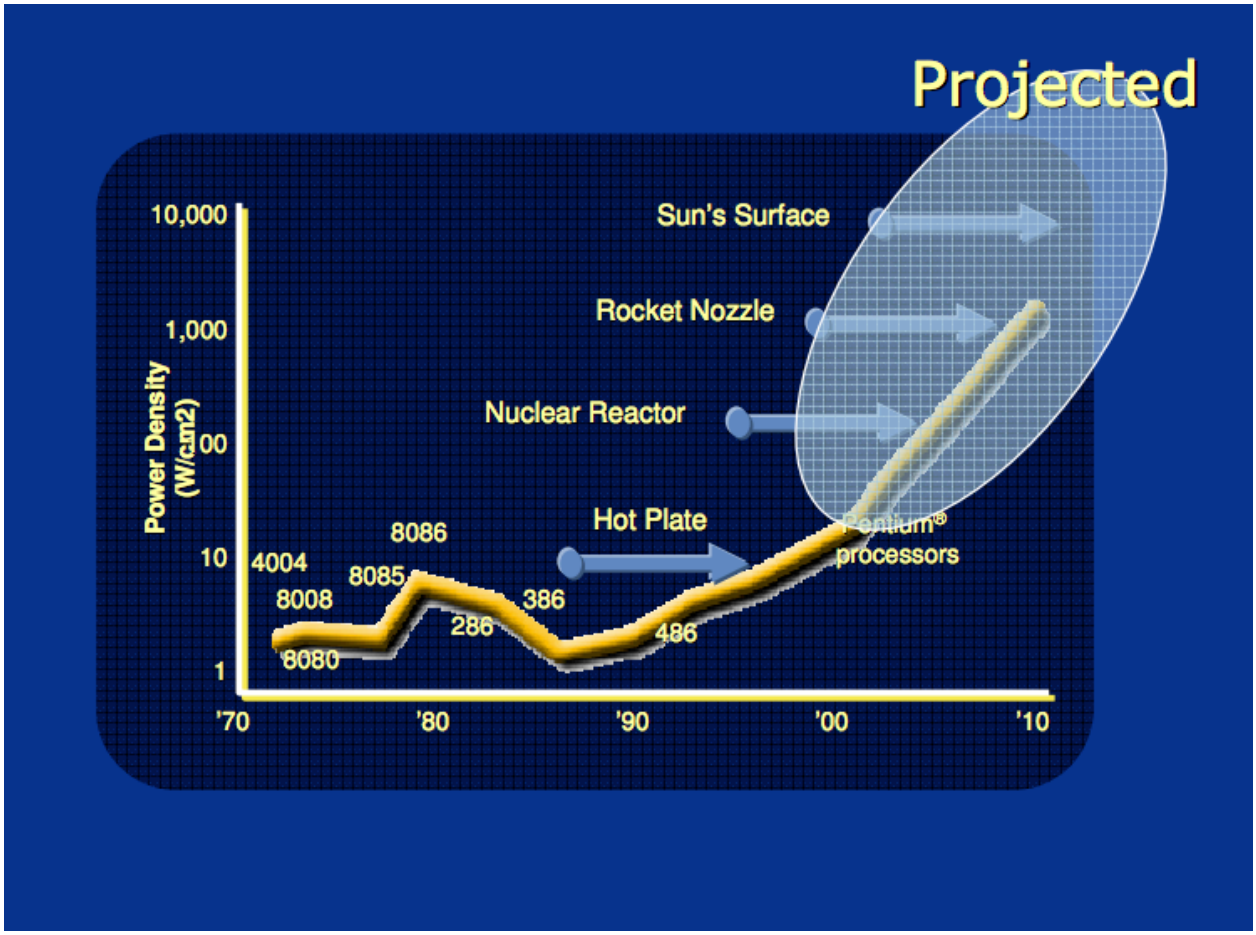
بر اساس قانون مور [۱] چگالی ترانزیستورهای تراشه‌های نیمه‌رسانا [۲] پس از گذشته به طور تقریبی هر هجدهماه، دو برابر می‌شود. در نتیجه می‌توان گفت که پردازنده‌ها کوچکتر، چگال‌تر و قدرتمندتر می‌شوند. بر اساس این قانون حداکثر فرکانس کاری پردازنده‌ها نیز قابل افزایش به نظر می‌رسد. اما به دلایل گوناگون روند افزایش فرکانس کاری پردازنده‌ها در سال‌های اخیر با کندی مواجه شده است. با ادامه روند افزایش فرکانس پردازنده‌ها، پیش‌بینی می‌شد که تا حوالی سال ۲۰۰۵ میلادی چگالی توان تراشه‌ها به سطح راکتورهای هسته‌ای برسد.

۱.۱ پردازنده‌های چندهسته‌ای

علی‌رغم پیشرفت چشمگیر قدرت محساباتی سخت‌افزارها و نیز نزدیک شدن به موانع عملی و فیزیکی برای افزایش فرکانس‌کاری آن‌ها، نیاز به بهبود عملکرد [۳] تراشه‌ها برای پشتیبانی از نیازمندی‌های جدید نرم‌افزار (به طور خاص رابط کاربری گرافیکی) و نیز انجام‌ها پردازش رو مجموعه‌داده‌های [۴] گسترده احساس می‌شد. در حدود سال ۲۰۰۰ میلادی اینتل با معرفی پردازنده‌های با معماری NetBurst انتظار دستیابی به فرکانس کاری 10Ghz را داشت اما در عمل به علت مشکلات گرما و توان مصرفی عملیات این چیپ‌ها فرکانس‌های بالای 4Ghz بدون استفاده از سیستم‌های خنک‌کننده بزرگ و پیچیده (معمولا مبتنی بر آب) امکان‌پذیر نبود.

در این میان ایده پردازنده‌های چندهسته‌ای به عنوان راه‌حلی برای افزایش کارایی ارائه شد. با توجه به اینکه در این فرکانس کاری پردازنده افزایش نمیابد، پارامترهای توان مصرفی و در نتیجه چگالی توان نیز تغییر شگرفی نمی‌کنند ولی چنانچه محسابات را بتوان به قسمت‌هایی با قابلیت موازی‌سازی

[۱] Moore's Law
[۲] Semiconductor
[۳] Performance
[۴] Dataset



شکل ۱.۱: پیش‌بینی رشد چگالی توان پردازنده‌ها

تقسیم کرد می‌توان زمان اجرا را به طور قابل توجهی کاهش داد. میزان ایده‌آل بهبود عملکرد محاسبه در این حالت از قانون امدال [۵] پیروی می‌کند.

لازم به ذکر است که تا قبل از ظهور پردازنده‌های چندهسته‌ای با تعریف امروزی گاها از چند پردازنده‌ی مرکزی مستقل برای انجام محاسبت استفاده می‌شد. در این معماری هر پردازنده عملا یک چیپ جداگانه بود که به واسطه یک باس [۶] مشترک به مادربورد و حافظه اصلی متصل می‌شد. این استقلال فیزیکی پردازنده‌ها مشکلات مختلفی ایجاد می‌کرد که در معماری‌های جدیدتر با قراردادن چند هسته [۷] پردازشی داخل یک تراشه تا حدی برطرف شده است.

[۵] Amdahl's law
[۶] Bus
[۷] Core

۲.۱ پردازنده‌های گرافیکی

تاریخچه پردازنده‌های گرافیکی به حدود سال‌های دهه ۷۰ میلادی بازمی‌گردد، زمانی که واحدهای سخت‌افزاری جداگانه برای بهبود عملکرد رایانه در اجرای بازی‌ها استفاده می‌شد. نسخه‌های اولیه چنین پردازنده‌هایی چیزی عملا گسترش جزیی معماری پردازنده‌های برداری [۸] برای کاربردهای گرافیکی بودند. در چنین پردازنده‌هایی که به طور خاص برای پردازش سیگنال و داده‌های در قالب ماتریس و آرایه طراحی شده بودند، تعداد زیادی واحد ریاضیاتی [۹] به طور همزمان دستورات یکسانی را روی قسمت‌های مختلف داده ورودی اجرا می‌کردند. این رویکرد که به اصطلاح مدل *دستور واحد و داده‌های متفاوت* [۱۰] نامیده می‌شود به طور خاص در محاسبات جبر خطی [۱۱] سودمند است.

پردازنده‌های گرافیکی در ابتدا سخت‌افزارهایی با عملکرد ثابت [۱۲] بودند و امکان برنامه‌پذیری [۱۳] نداشتند. با افزایش قدرت پردازنده‌های گرافیکی طی نسل‌های متمادی و آشکار شدن پتانسیل این روش پردازش برای کاربردهایی خارج از حوزه گرافیک و بازی‌های رایانه‌ای، به مرور زمان قابلیت برنامه‌پذیری نسبی به این سخت‌افزارها اضافه شد و امروزه واسط‌های نرم‌افزاری سطح بالای قدرتمندی مانند کودا [۱۴] توسعه داده توسط شرکت Nvidia و OpenCL برای منظور پیاده‌سازی برنامه‌های موازی عام‌منظوره روی این تراشه‌ها در دسترس هستند.

یک پردازنده گرافیکی به طور معمول متشکل است از تعدادی چندپردازنده [۱۵] که به صورت موازی دستورات (نه لزوما یکسان) را اجرا می‌کنند. به عنوان مثال پردازنده گرافیکی Nvidia Tesla K40 بر پایه معماری Kepler از ۱۵ چندپردازنده جریانی [۱۶] هر کدام با ۱۹۲ هسته پردازشی عدد طبیعی [۱۷] ، ۶۴ هسته پردازشی عدد ممیزدار [۱۸] و ۳۲ واحد انتقال داده [۱۹] تشکیل یافته است. این هسته‌ها به ۱۲ مجموعه *دستور واحد و داده‌های متفاوت* تقسیم می‌شود که هریک می‌توانند به طور مستقل دستور واحدی را روی داده ورودی خود اجرا کند . همچنین هر چندپردازنده جریانی دارای یک واحد

[۸] Vector Processor
[۹] Arithmetic Logic Unit
[۱۰] Single Instruction, Multiple Data (SIMD)
[۱۱] Linear Algebra
[۱۲] Fixed-Function
[۱۳] Programmability
[۱۴] CUDA: Compute Unified Device Architecture
[۱۵] Multiprocessor
[۱۶] Streaming Multiprocessor (SM)
[۱۷] Integer
[۱۸] Floating Point
[۱۹] Load/Store Unit

حافظه مشترک [۲۰] میان تمام هسته‌های پردازشی خود است که برای ذخیره داده‌های مورد نیاز در پردازش فعلی و عدم نیاز به بارگیری [۲۱] آن‌ها از حافظه اصلی در حین اجرا استفاده می‌شود.

در چنین معماری که با نام *دستور واحد و ریسه‌های متعدد* [۲۲] بازاریابی می‌شود تعداد زیادی ریسه به طور همزمان روندهای اجرایی متفاوتی را روی داده اعمال می‌کنند. یک پردازنده گرافیکی با معماری کپلر می‌تواند ۲۸۸۰ ریسه را به صورت موازی اجرا کند.

شکل ۲.۱: ساختار داخلی یک چندپردازنده جریانی در معماری Kepler

به لطف وجود چنین ظرفیت موازی‌سازی گسترده‌ای در پردازنده‌های گرافیکی زمان اجرای محاسبات روی آن‌ها به شکل قابل توجهی کاهش می‌یابد. اما به دلایل مختلف مانند وابستگی‌داده‌ای

[۲۰] Shared Memory
[۲۱] Fetch
[۲۲] Single Instruction, Multiple Threads (SIMT)

بین دستورات، عدم قابلیت موازی‌سازی همه بخش‌های پردازش، رفتارهای مختلف [۲۳] ریسه‌ها و در دستورات تصمیم‌گیری [۲۴] و همچنین تاخیرهای دسترسی به حافظه کارای پردازنده گرافیکی در شرایط واقعی بسیار پایین‌تر از مقدار آن روی کاغذ است.

۳.۱ ساختار پایان‌نامه

در ادامه و پس از پایان مقدمه، در فصل دوم مفاهیم پایه‌ای در طراحی پردازنده‌های چندهسته‌ای و به خصوص پردازنده‌های گرافیکی و به خصوص ساختاربندی حافظه در این تراشه‌ها را بررسی می‌کنیم. فصل سوم مروری بر کارهای پیشین در موضوع پایان‌نامه خواهد بود. در فصل چهارم شهود و انگیزه‌های اولیه برای این مطالعه را بررسی می‌کنم فصل پنجم را به معرفی شبیه‌ساز و بررسی نتایج حاصل از شبیه‌سازی اختصاص می‌دهیم. در نهایت در فصل ششم به جمع‌بندی و بررسی کارهای آتی می‌پردازیم.

[۲۳] Divergence
[۲۴] Decision Making

۲ مفاهیم پایه

در این فصل ابتدا مروری کلی بر مفاهیم پردازنده‌های چندهسته‌ای و دلایل روی‌آوردن به این پردازنده‌ها خواهیم داشت. در ادامه به بیان کلیاتی در مورد پردازنده‌های گرافیکی و معماری و مدل برنامه‌سازی آن‌ها خواهیم پرداخت.

۱.۲ دلایل روی آوردن به پردازنده‌های چندهسته‌ای

زمانی که اندازه مشخصه [۱] تزانزیستورها با فاکتور k کاهش می‌یابد، به دلیل کوتاه‌تر شدن سیم‌ها و کاهش اندازه خازن در گیت‌ها [۲] ، فرکانس کلاک [۳] نیز با فاکتور k قابل افزایش است. همچنین تعداد ترانزیستورهای موجود در واحد سطح با فاکتور x^2 و اندازه قالب [۴] ترازیستورها نیز با فاکتور k قابل افزایش می‌یابد. در چنین شرایطی قدرت پردازشی نیز به صورت تئوری با فاکتور k^4 افزایش می‌یاد. هرچند در عمل به دلیل مواردی مانن توازی پنهان [۵] یا رفتار غیرقابل پیش‌بینی حافظه نهان x^3 این فاکتور به طور عملی در مرتبه x^3 افزایش می‌یابد. به این نسبت‌ها قانون دانار گفته می‌شود [۶] . با این اوصاف به نظر می‌رسد که با معرفی هر نسل جدید پردازنده‌ها با اندازه مشخه کوچکتر باید شاهد بهبود شگرف در عملکرد نرم‌افزارها باشیم. اما در عمل این رشد با موانعی روبه‌روست که در ادامه به آن‌ها می‌پردازیم.

---

[۱] Feature Size
[۲] Gate
[۳] Clock
[۴] Die
[۵] $HiddenParallelism$
[۶] Dannar Scaling (MOSFET Scaling)

۱.۱.۲ چگالی توان

چگالی توان [۷] به شکل میزان توان (نرخ انتقال انرژی) بر واحد حجم تعریف می‌شود. میزان مصرف توان در تراشه‌ها به نرخ تغییر وضعیت گیت‌ها، یعنی نرخی که در آن خروجی یک گیت از صفر به یک تغییر می‌کند، بستگی‌دارد. به این دلیل به اصطلاح گفته می‌شود که تراشه‌ها نرخ توان مصرفی پویا دارند. با توجه به توضیح فوق انتظار می‌رود که نرخ توان مصرفی با افزایش فرکانس به طور خطی افزایش پیدا کند. با توجه به افزایش نمایی فرکان پردازنده‌ها در سال‌های پایانی دهه ۹۰ و اوایل قرن ۲۱ام، انتظار می‌رفت چگالی توان پردازنده‌ها در صورت حفظ این نرخ رشد تا سال ۲۰۱۰ میلادی به ۱۰۰۰۰ وات بر سانتی‌متر مربع یعنی چیزی در حدود چگالی توان در سطح خورشید برسد. واضح است که تراشه‌های نیمه رسانا در چنین وضعیتی تبخیر خواهند شد.

۲.۱.۲ دیوار حافظه

به طور معمول هر دسترسی به حافظه اصلی [۸] در حدود صدها سیکل کلاک زمان می‌برد. به طور مثال پردازنده ممکن است برای اجرای محاسبه‌ای که چند کلاک طول بکشد صدها کلاک منتظر دریافت داده و نوشتن مجدد آن در حافظه بماند. به طور میانگین در سال‌های گذشته فرکانس حافظه هر شش سال دوبرابر می‌شد در حالی‌که که در تبعیت از قانون مور فرکانس پردازنده هر دو سال دو برابر می‌شد. این تفاوت در نرخ رشد سبب ایجاد یک شکاف بزرگ میان عملکرد پردازنده و حافظه می‌شود و حافظه را به گلوگاهی [۹] برای عملکرد سیستم تبدیل می‌کند و تاثیر افزایش فرکانس پردازنده را به شدت کاهش می‌دهد. معماران سخت‌افزار با بهره‌گیری از ایده‌ها و روش‌های مختلف از جمله استفاده از چندین لایه حافظه نهان و بهینه سازی‌هایی مانند *بارگذاری با تاخیر* [۱۰] و *کپی هنگام نوشتن* [۱۱] سعی در مخفی کردن اثر سرعت حافظه از دید پردازنده دارند اما در نهایت مشکل همچنان باقی است. شکل ۲.۲ شکاف بین عملکرد پردازنده و حافظه اصلی را در سال‌های گذشته نشان می‌دهد.

[۷] Power
[۸] Random Access Memory (RAM)
[۹] Bottleneck
[۱۰] Lazy Writeback
[۱۱] Copy on Write

۲.۲ پردازنده‌های چندهسته‌ای به عنوان یک راه‌حل

در زمانی که افزایش فرکانس پردازنده‌ها دیگر ممکن به نظر نمی‌رسد، مهندسان ایده استفاده از چند پردازنده روی یک تراشه را برای بهبود عملکرد مطرح کردند. با توجه به اینکه کارایی یک پردازنده با فرکانس کاری و تعداد هسته‌های آن متناسب است، با افزایش تعداد هسته و ثابت نگه داشتن فرکانس می‌توانیم به عملکرد بهتری برسیم. با پذیرفته شدن این معماری توسط تولید کنندگان مطرح مانند Intel و AMD از آن به بعد:

- چگالی ترانزیستورها می‌تواند مانند قبل هر دو سال دو برابر شود
- فرکانس پردازنده‌ها افزایش نمیابد (بعضا شاهد کاهش فرکانس برای ملاحظات توان مصرفی هستیم)
- به جای دو برابر کردن فرکانس تمرکز روی دو برابر کردن تعداد هسته‌های پردازشی است

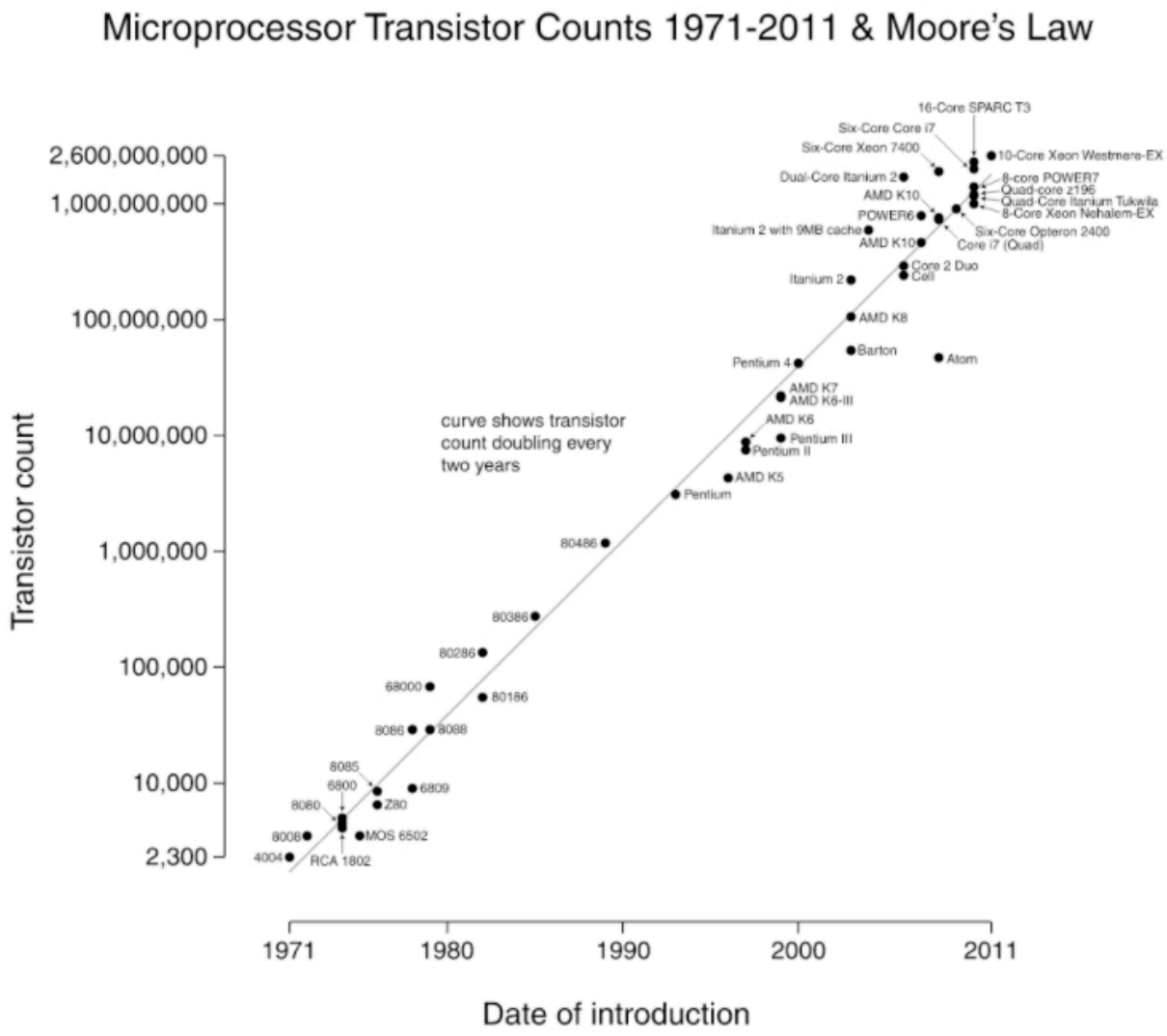
۳.۲ پردازنده‌های گرافیکی عام‌منظوره

به پردازنده‌های گرافیکی که قابلیت برنامه‌ریزی داشته باشد پردازنده گرافیکی عام‌منظوره [۱۲] گفته می‌شود. امروزه عمده کاربرد این پردازنده‌ها در محاسبات سنگین، شکستن رمزها، ارزهای رمزنگاری‌شده [۱۳] و شبیه‌سازی‌های علمی است.

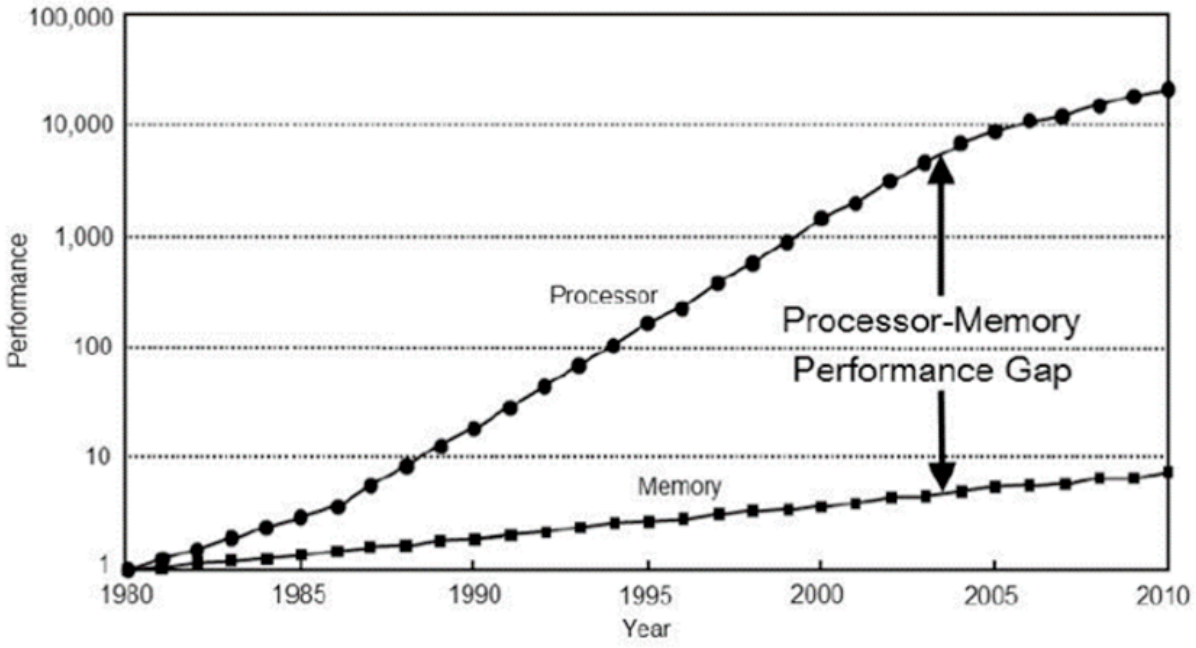
بر خلاف پردازنده‌های مرکزی که برای اجرای سیستم عامل و سوییچ کردن [۱۴] بین تعداد زیادی پردازه [۱۵] اجرا و پنهان کردن تاخیرهای حافظه برای حفظ پاسخگویی حداکثری طراحی شده‌اند، پردازنده‌های گرافیکی با هدف حداکثر سرعت در محاسبات تولید می‌شوند و بسیاری از پیچیدگی‌های داخلی پردازنده مرکزی را از معماری خود حذف می‌کنند. شکل ۴.۲ شکاف بین عملکرد این دو سخت‌افزار را در محاسبات روی اعداد ممیزدار نشان می‌دهد.

۴.۲ استفاده از پردازنده گرافیکی برای مسائل عام‌منظوره

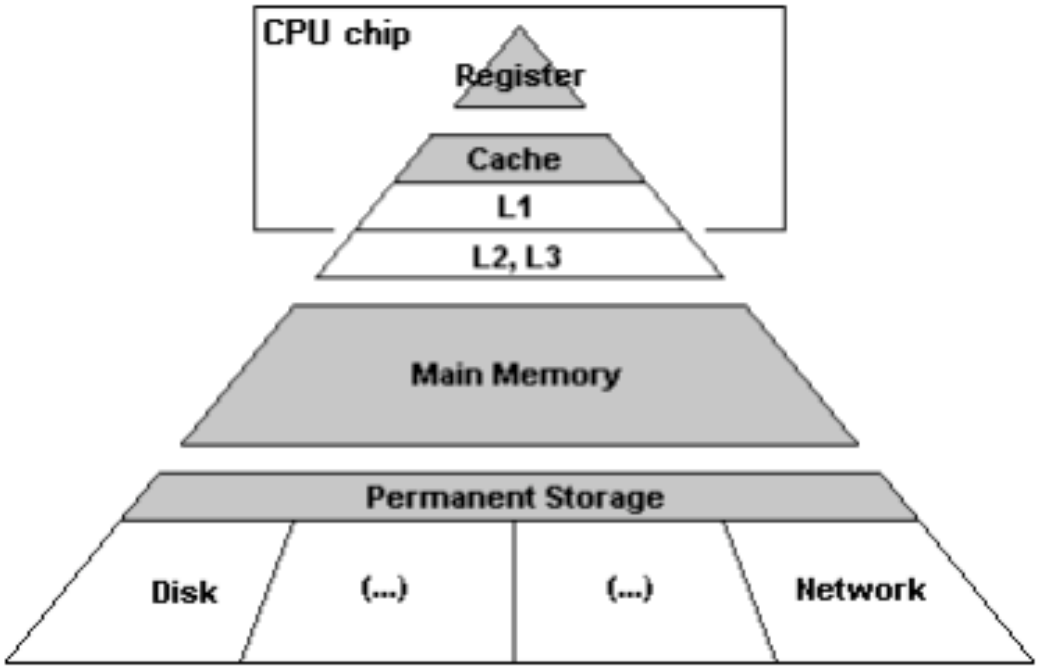
[۱۲] General Purpose Graphics Processing Unit
[۱۳] Cryptocurrency
[۱۴] Context Switching
[۱۵] Process



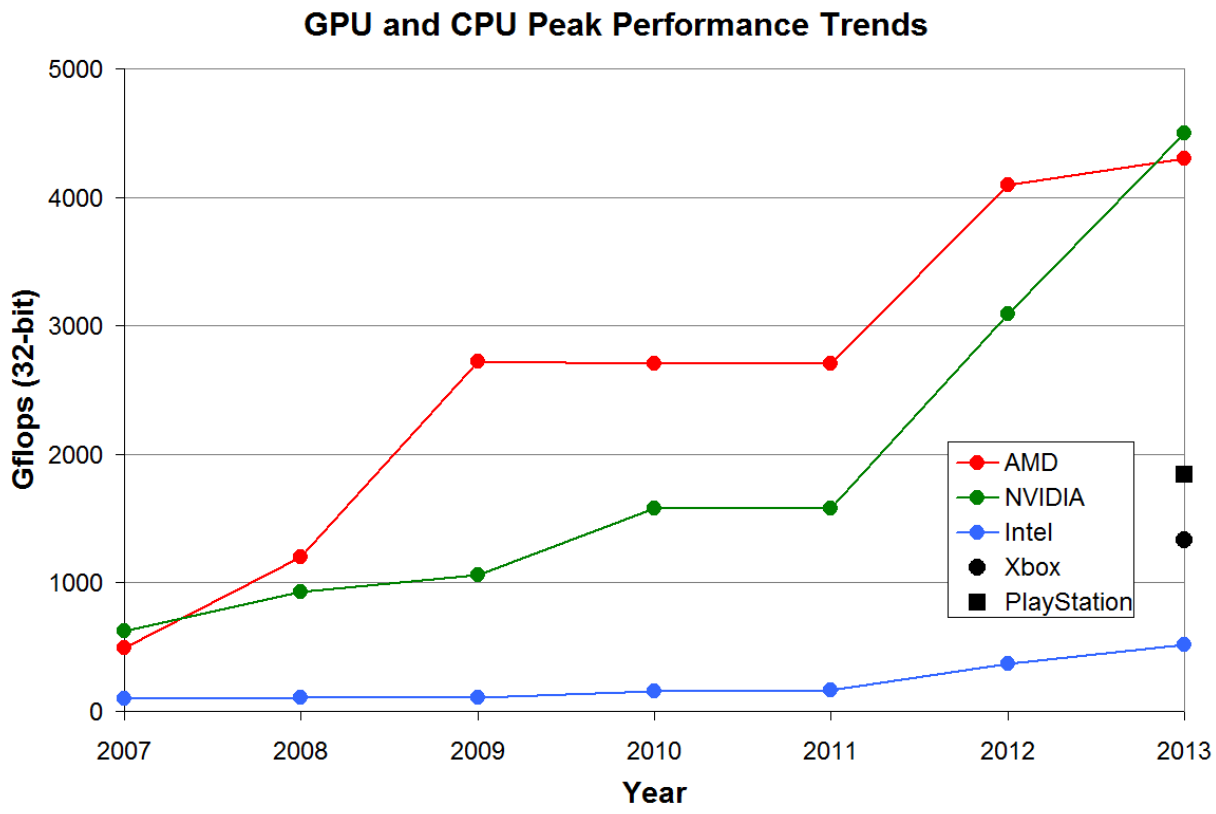
شکل ۱.۲: تبعیت رشد تعداد ترانزیستورها از قانون مور



شکل ۲.۲: شکاف بین عملکرد حافظه و پردازنده



شکل ۳.۲: سلسله مراتب حافظه در یک پردازنده امروزی



شکل ۴.۲: شکاف بین عملکرد پردازنده مرکزی و پردازنده گرافیکی در محاسبات عددی

۳ کارهای پیشین

در این فصل به طور اجمالی به بررسی مطالعات انجام شده روی موضوع این پایان‌نامه می‌پردازیم.

از آن‌جا که تاثیر حافظه مشترک در محدود کردن عملکرد پردازنده گرافیکی، به خصوص موازات سطح ریشه و تعداد دستورالعمل در کلاک به اندازه‌ی سایر عوامل از جمله رجیستر فایل، حافظه نهان و واگرایی پرش نیست، مطالعاتی محدودی روی این موضوع انجام شده است و مقالات کمی را می‌توان یافت که ایده‌های جدید در این زمینه مطرح کنند.

از جمله مقالات چند سال گذشته در مورد تاثیر حافظه مشترک و تاثیر آن عملکرد پردازنده‌های گرافیکی می‌توانی به [۱۱] ، [۱۲] ، [۱۳] و [۱۴] اشاره کرد. این مقالات بیشتر به مبحث عدم استفاده مناسب [۱] از حافظه مشترک به علت تلاقی دسترسی به بانک‌های حافظه و وجود متغیرها و داده‌های مشترک بین چند ریسه در آن اشاره کرد.

رویکرد ما در این پایان‌نامه بررسی تاثیر حافظه مشترک در بارهای کاری محاسباتی عام‌منظوره و امکان جایگزینی این حافظه با راهکاری بهینه‌تر در سخت‌افزارهای طراحی شده برای محاسبات سرعت بالا بوده است. چنین دیدی هرگز در مطالعات انجام شده تا به امروز مورد توجه قرار نگرفته است، از این رو امیدواریم توانسته باشیم ایده‌های جدید وارد این حوزه کنیم.

[۱] Underutilization

۴ انگیزه و شهود

در این فصل ابتدا به بررسی وظایف حافظه مشترک و کاربردهای آن در پردازش‌های مختلف می‌پردازیم و سپس روش‌های پیشنهادی خود را برای اندازه‌گیری عملکرد آن تحت بارهای کاری علمی و نتایج به دست آمده را بررسی می‌کنیم.

در ادامه روشی برای بهبود عملکرد کلی تراشه گرافیک با تکیه بر تغییر معماری حافظه مشترک پیشنهاد می‌دهیم و شهودی برای تاثیرگذار بودن این رویکرد ارائه می‌کنیم.

۱.۴ حافظه چرک‌نویس

حافظه چرک‌نویس [۱] به حافظه‌ای اطلاق می‌شود که نتایج میانی محاسبات را نگهداری می‌کند. scratchpad معمولا نزدیک‌ترین واحد حافظه به ALU پس از رجیسترهاست و قادر به دسترسی مستقیم به حافظه اصلی [۲] است. از آنجا که عمده نتایج میانی در محاسبات سنگین در پایان دور ریخته می‌شوند استفاده از حافظه اصلی (و به تبع آن حافظه نهان) برای ذخیره‌سازی آن‌ها به علت سرعت کم و نیز احتمال تاثیر منفی بر سایر دستورات در حال اجرا (مصرف حافظه نهانی که می‌توانست به آن‌ها اختصاص پیدا کند) ضرورتی ندارد و در عوض از یک حافظه سریع‌تر داخلی به این منظور استفاده می‌شود.

مزیت دیگر این واحد حافظه زمان دسترسی قابل پیش بینی به آن است، زیرا داده قبل از رسیدن به پردازشگر از لایه‌های حافظه نهان عبور می‌کند و در زمان ثابتی قابل دسترسی است.

[۱] Scratchpad Memory
[۲] Direct Memory Access (DMA)

۲.۴ حافظه مشترک

در پردازنده‌های مبتنی بر معماری کودا، به هر چندپردازنده جریانی حافظه‌ای به عنوان *حافظه مشترک* اختصاص داده می‌شود که عملا نقش همان چرک‌نویس را بازی می‌کند. برنامه‌نویس می‌تواند از این حافظه برای ذخیره نتایج میانی و وضعیت فعلی پردازش و یا به عنوان یک کپی سریع‌تر از حافظه اصلی استفاده کند. به عنوان مثال در معماری فرمی هر چندپردازنده دارای یک واحد حافظه ۶۴ کیلوبایتی است که بستگی به نیاز می‌تواند ۱۶ یا ۴۸ کیلوبایت از آن را در ابتدای اجرای کرنل به عنوان حافظه مشترک استفاده کند.

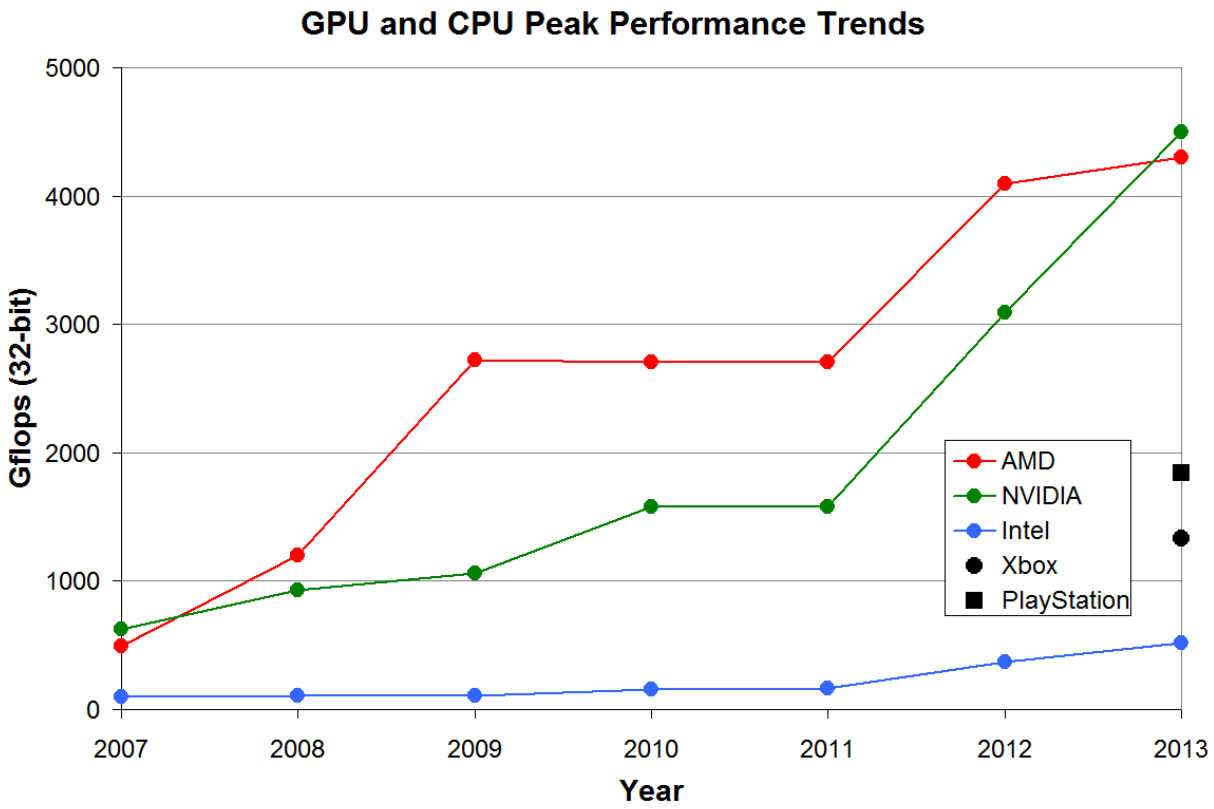
۳.۴ شبیه‌ساز GPGPUSim

GPGPUSim یک شبیه‌ساز نوشته شده به زبان C++ برای شبیه‌سازی عملکرد پردازنده‌های گرافیکی است. با اینکه تاکید اصلی در طراحی این شبیه‌ساز برای مطابقت آن با معماری CUDA بوده است، اما این شبیه‌ساز به صورت داخلی از مدل انتزاعی قابل انعطافی استفاده می‌کند که می‌تواند در صورت لزوم برای شبیه‌سازی سخت‌افزارهای دیگر تولیدکنندگان هم مورد استفاده قرار بگیرد. این شبیه‌ساز نرخ دستور بر کلاک [۳] تقریبا یکسان (با همبستگی حدود ۹۸ درصد با سخت‌افزار کودا) نشان می‌دهد و به از این جهت برای سنجش عملکر پردازنده گرافیکی بسیار مناسب به نظر می‌رسد. شکل ۱.۴ این رفتار شبیه‌ساز را نشان می‌دهد . بحث بیشتر در مورد جزییات داخلی این شبیه‌ساز خارج از حوصله این نوشتار است.

۱.۳.۴ ثبت دسترسی‌ها به حافظه در شبیه‌ساز

برای ثبت دسترسی‌ها به حافظه لازم است تغییرات را داخل کد شبیه‌ساز اعمال کنیم. فایل *instruc-tions.cc* در پوشه *cuda-sim* شامل کدهای مربوط به مدیریت دسترسی‌های حافظه است. روند اجرای تمام دستورات خواندن [۴] و نوشتن [۵] به سطوح مختلف حافظه، چه به عنوان دستور مستقل و چه به عنوان بخشی از یک دستور دیگر (مانند دسترسی به ثبات‌ها) از توابع تعریف شده داخل این فایل می‌گذرد. موارد زیر را برای هر دسترسی ثبت می‌کنیم:

[۳] Instructions Per Clock (IPC)
[۴] Read
[۵] Write



شکل ۱.۴: همبستگی نرخ IPC میان شبیه‌ساز و سخت‌افزار نسل Fermi

- نوع دسترسی که می‌تواند خواندن یا نوشتن باشد،
- آدرس مورد دسترسی
- تعداد بایت مورد دسترسی
- زمان دسترسی (شماره کلاک)
- شناسه کرنل
- شناسه ریسه
- شناسه CTA

۴.۴ بارهای کاری مورد استفاده

در طول این مطالعه برای بررسی عملکر حافظه مشترک از ۳ مجموعه بنچمارک [۶] استفاده شده است:

۱. مجموعه بنچمارک رودینیا [۷] دانشگاه ویرجینا

۲. مجموعه بنچمارک پاربویل [۸] دانشگاه ایلینویز

۳. مجموعه بنچمارک ISPASS

این بنچمارک‌ها مجموعه‌ای از الگوریتم‌های رایج در حوزه‌های مختلف علمی و مهندسی را پوشش می‌دهند که در جداول زیر به بررسی آن‌ها می‌پردازیم.

۱.۴.۴ دسترسی به حافظه مشترک در بارهای کاری مورد بررسی و انگیز اولیه

با اجرای شبیه‌ساز روی ۳۲ بار کار موجود در مجموعه سه بنچمارک مشاهده می‌شود که تنها ۱۸ بار کاری در طی روند اجرای خود از حافظه مشترک استفاده می‌کنند.

با توجه به دامنه گسترده بارهای کاری مورد آزمون می‌توان ادعا کارد که این بنچمارک‌ها طیف کاملی از کاربردهای عام منظوره رو پوشش می‌دهند. مشاهدات فوق انگیزه اولیه برای ادامه این مطالعه را ایجاد می‌کند. حافظه مشترک در بسیاری از محاسبات عملا توسط برنامه‌نویس مورد استفاده قرار نمی‌گیرد. با توجه به میزان قابل توجه حافظه مشترک یافتن راه حلی برای استفاده بهینه از آن مستقل

[۶] Benchmark
[۷] Rodinia
[۸] Parboil

Benchmark	Summary	Application
BFS	Breadth-First Search	Graph Algorithms
CUTCP	Distance-Cutoff Coulombic Potential	Molecular Simulation
HISTO	Saturating Histogram	Statistics
LBM	Lattice-Boltzmann Method	Fluid Dynamics
MM	Dense Matrix-Matrix Multiply	Linear Algebra
MRI-GRIDDING	Magnteic Resonance Imaging - Gridding	Medical Imaging
MRI-Q	Magnetic Resonance Imaging - Q	Medical Imaging
SAD	Sum of Absolute Differences	MPEG video encoding
SPMV	Sparse-Matrix Dense-Vector Multiplication	Linear Algebra
SGEMM	Matrix-Matrix Multiplication	Linear Algebra
STENCIL	3-D Stencil Operation	Graphics
TPACF	Two Point Angular Correlation Function	Atronomy

جدول ۱.۴: گستره بنچمارک‌های موجود در پکیج پاربویل

از نوع بارکاری در حال اجرا می‌تواند عملکرد پردازنده را بهبود بخشد. برای محسابه بهبود عملکرد پردازنده از معیار IPC استفاده خواهیم.

تخصیص و آزادسازی فضا روی حافظه مشترک توسط برنامه‌نویس و صریحا انجام می‌شود که این امر می‌تواند پیچیدگی‌هایی را به همراه داشته باشد. از زمان ظهور زبان‌های مدرن برنامه‌سازی، مدیریت حافظه به طور سنتی توسط کامپایلر و یا محیط اجرا [۹] انجام می‌شده و به خصوص در زبان‌هایی که بعد از C++ وارد بازار شده‌اند، تدابیر مختلفی برای پنهان کردن جزییات روند مدیریت حافظه از دید برنامه نویس اندیشیده شده است. در داده‌ساختارهای رایج مورد استفاده مانند آرایه، بردار، ماتریس، مجموعه بیت و بعضا لیست‌ها [۱۰] داده‌ها به شکل دنبال هم در حافظه ذخیره می‌شوند و از این رو در

[۹] Runtime Environment

[۱۰] چنانچه پیاده‌سازی به شکل آرایه پویا باشد

Benchmark	Summary	Application
Kmeans	Dense Linear Algebra	Data Mining
Needleman-Wunsch (NW)	Dynamic Programming	Bioinformatics
HotSpot (HS)	Structured Grid	Physics Simulation
Back Propagation (BP)	Unstructured Grid	Pattern Recognition
SRAD	Structured Grid	Image Processing
Leukocyte Tracking (LC)	Structured Grid	Medical Imaging
Breadth-First Search (BFS)	Graph Traversal	Graph Algorithms
Stream Cluster (SC)	Dense Linear Algebra	Data Mining
MUMmer (MUM)	Graph Traversal	Bioinformatics
CFD Solver (CFD)	Unstructured Grid	Fluid Dynamics
LU Decomposition (LUD)	Dense Linear Algebra	Linear Algebra
Heart Wall Tracking (HW)	Structured Grid	Medical Imaging

جدول ۲.۴: گستره بنچمارک‌های موجود در پکیج رودینیا

دسترسی به آن‌ها به طور طبیعی از هم‌مکانی [۱۱] بالایی برخوردار است که این مورد می‌تواند عملکرد حافظه نهان را به شکل قابل توجهی افزایش دهد. به طور مثال در حوزه پردازش تصویر داده‌های اولیه معمولا مجموعه‌ای از پیکسل‌ها هستند که به شکل طبیعی در قالب ماتریس ذخیره‌سازی می‌شوند و یا در حوزه بایوانفورماتیک داده‌های مورد پردازش معمولا توالی‌های طولانی ژنومی همراه با متاداده‌های اضافه‌شده به آن‌ها هستند. چنین داده ساختارهای ترتیبی‌ای [۱۲] به خوبی توسط سخت‌افزار مدیریت می‌شوند.

به علاوه وجود یک حافظه مستقل از RAM و استفاده صریح از آن توسط برنامه‌نویس، ساختار کد را به معماری سخت‌افزار و محدودیت‌ها آن وابسته می‌کند. همچنین با توجه به کارکرد مشخص حافظه مشترک، چنانچه ساختار مساله نیازی به وجود آن نداشته باشد، تعداد قابل توجهی ترانزیستور در طی روند اجرا بی استفاده می‌مانند و تنها توان مصرفی چیپ و گرمای تولیدشده توسط آن‌را افزایش

[۱۱] Locality
[۱۲] Sequential

Benchmark	Summary
AES	Cryptography
Breadth-First Search (BFS)	Graph Traversal Algorithm
CP	N/A
DG	Discontinuous Galerkin Solvers
LPS	3D Laplace Discretisation
LIB	N/A
MUM	High-Throughput Genome Sequence Alignment
NN	Neural Network
NQU	N-Queens Solver
RAY	Ray Tracing
STO	dDistributed storage systems
WP	Numerical Weather Prediction

جدول ۳.۴: گستره بنچمارک‌های موجود در پکیج ISPASS

Benchmark	Workloads
Parboil	BFS, CUTCP, HISTO, MRI-GRIDDING, SAD, SGEMM, STENCIL
Rodinia	HW, HS, LMD, NW, BP, PF
ISPASS	LPS, NQU, STO

جدول ۴.۴: لیست بارهای کاری با دسترسی به حافظه مشترک

می‌دهند.

انگیزه اصلی ما در این پژوهش پیدا کردن راه حلی برای جایگزینی حافظه مشترک با حافظه‌ای چندمنظوره و تا حد امکان پنهان از دید برنامه‌نویس و در عین حال مطابقت با کدهای نوشته‌شده با فرض وجود آن است.

۵.۴ معماری جایگزین پیشنهادی

حافظه مشترک از نظر سخت‌افزاری ویژگی چندان خاصی ندارد به طوری که در پردازنده‌های جدیدتر خانواده کودا امکان تقسیم حافظه اختصاص‌یافته به هر چندپردازنده جریانی بین حافظه نهان و حافظه

مشترک وجود دارد. با توجه به اینکه کارایی حافظه‌نهان از دید برنامه‌نویس پنهان [۱۳] است و به علاوه افزایش آن همواره می‌تواند عملکرد سخت‌افزار را بهبود بخشد به نظر می‌رسد که راه حل باید به نوعی شامل جایگزینی حافظه مشترک با حافظه نهان باشد. با توجه به اینکه در نظر داریم معماری پیشنهاد ما همچنان با کدهای موجود سازگار باشد، انتقال حافظه مشترک از روی هر چندپردازنده به حافظه DRAM و نگاشت فضای آدرس‌دهی آن به طور مجازی ایده‌آل به نظر می‌رسد. برای جبران تاخیر در دسترسی به DRAM زمانی که کد نیاز به استفاده از حافظه مشترک داشته باشد قسمتی از حافظه هر چندپردازنده به عنوان حافظه نهان برای فضای آدرس‌دهی مشترک عمل خواهد کرد. بنابراین کلیت معماری پیشنهادی ما و مزیت‌های مورد انتظار آن به شرح زیر است:

تغییر	مزیت
حذف حافظه مشترک از هر ریزپردازنده و جایگزینی آن با حافظه نهان	۱. عدم بی‌استفاده ماندن حافظه نهان و مصرف توان اضافی توسط آن در صورت عدم نیاز برنامه. ۲. بهبود عملکرد حافظه نهان به علت افزایش حجم آن
نگاشت فضای آدرس دهی مشترک به حافظه اصلی و استفاده از حافظه نهان هر چندپردازنده برای جبران تاخیر	۱. به علت حجم به نسبت کم حافظه مشترک، حافظه نهان مورد استفاده برای آن کوچک خواهد بود ۲. به علت پنهای‌باند بالای حافظه اصلی حافظه مشترک روی هم چندپردازنده دیگر گلوگاهی برای IPC نخواهد بود

جدول ۵.۴: کلیت معماری جایگزین پیشنهادی

در فصل بعد به تفصیل این معماری و مزایای آن را بررسی خواهیم کرد.

[۱۳] Transparent

۵ نتایج بررسی

در این بخش ابتدا تاثیر اعمال معماری پیشنهادی خود را بر روی عملکرد حافظه نهان و حافظه مشترک می‌سنجیم.

۱.۵ جمع آوری داده

با توجه به تغییرات اعمال شده در شبیه‌ساز، تما دسترسی‌ها حافظه برای ۱۸ بار کاری با دسترسی به حافظه مشترک را ثبت می‌کنیم. اجرای کامل این بارهای کاری بر روی سرور اصلی پژوهشکده IPM قریب به ۴ روز زمان می‌برد و در نهایت در حدود ۱۲۰۰ گیگابایت داده تولید می‌کند. این دسترسی‌ها به حافظه به ۵ دسته تقسیم‌بندی می‌شوند:

۱. حافظه مشترک

۲. حافظه RAM پردازنده میزبان

۳. حافظه DRAM پردازنده گرافیکی

۴. حافظه Texture پردازنده گرافیکی

۵. حافظه ثابت [۱] پردازنده گرافیکی

برای بررسی عملکرد حافظه نهان یک شبیه‌سازی حافظه نهان به زبان C++ توسعه داده شد. شبیه‌ساز از معماری Set Associative استفاده می‌کند و قابل پیکربندی مطابق با ویژگی‌های سخت‌افزار مورد نظر است. با توجه به دقت بالای شبیه‌سازی GPGPUSim برای معماری Fermi برای حافظه نهان نیز از پیکربندی پردازنده گرافیکی GTX۴۸۰ استفاده شد:

[۱] Constant

RAM	4GB
Word Size	128 bytes
Per SM Memory	96KB
Cache Page Size	16KB

برای سهولت در تحلیل نتایج کل زمان اجرای هر باری کاری را به پنجره‌های با طول ۱۰۰۰ کلاک تقسیم می‌کنیم و نتایج نرخ برخورد را برای هر پنجره گزارش می‌کنیم.

۲.۵ عملکرد حافظه نهان برای آدرس‌های حافظه اصلی پردازنده گرافیکی

با توجه به تکیه معماری پیشنهادی ما بر افزایش حافظه نهان و بهبودی عملکرد حاصل از آن باید نشان دهیم که با افزایش میزان حافظه نهان موجود در هر چندپردازنده می‌توانیم بهبود قابل توجهی در عملکرد آن، یعنی نرخ برخورد را شاهد باشیم. به این منظور دسترسی‌های مربوط به حافظه DRAM برای هر بار کاری با پیکربندی‌های مختلف به شبیه‌سازی ورودی می‌دهیم و نرخ برخورد را برای آن‌ها می‌سنجیم. انتظار داریم که با افزایش حجم حافظه نهان نرخ برخورد بهتری را شاهد باشیم.

جداول زیر نتایج به دست آمده را نشان می‌دهند. در طی این شبیه‌سازی اندازه هر صفحه [۲] حافظه نهان را ۱۶ کیلوبایت در نظر گرفته‌ایم.

Workload/Cache Size	32Kb	48Kb	64Kb	96Kb
BFS	0.252	0.349	0.425	0.489
LPS	0.599	0.777	0.805	0.811
NQU	0.937	0.934	0.960	0.971
MUM	0.295	0.310	0.318	0.321
NN	0.678	0.936	0.965	0.967
WP	0.579	0.645	0.679	0.695

جدول ۱.۵: درصد برخورد حافظه نهان برای آدرس‌های DRAM بارهای کاری بنچمارک ISPASS

Workload/Cache Size	32Kb	48Kb	64Kb	96Kb
Back Propagation	0.564	0.745	0.779	0.784
Heart Wall Track	0.255	0.476	0.678	0.822
Particle Filter	0.620	0.764	0.846	0.881
StreamCluster	0.264	0.347	0.354	0.360
Needleman-Wunsch	0.545	0.545	0.547	0.550
Gaussian	0.526	0.723	0.779	0.782

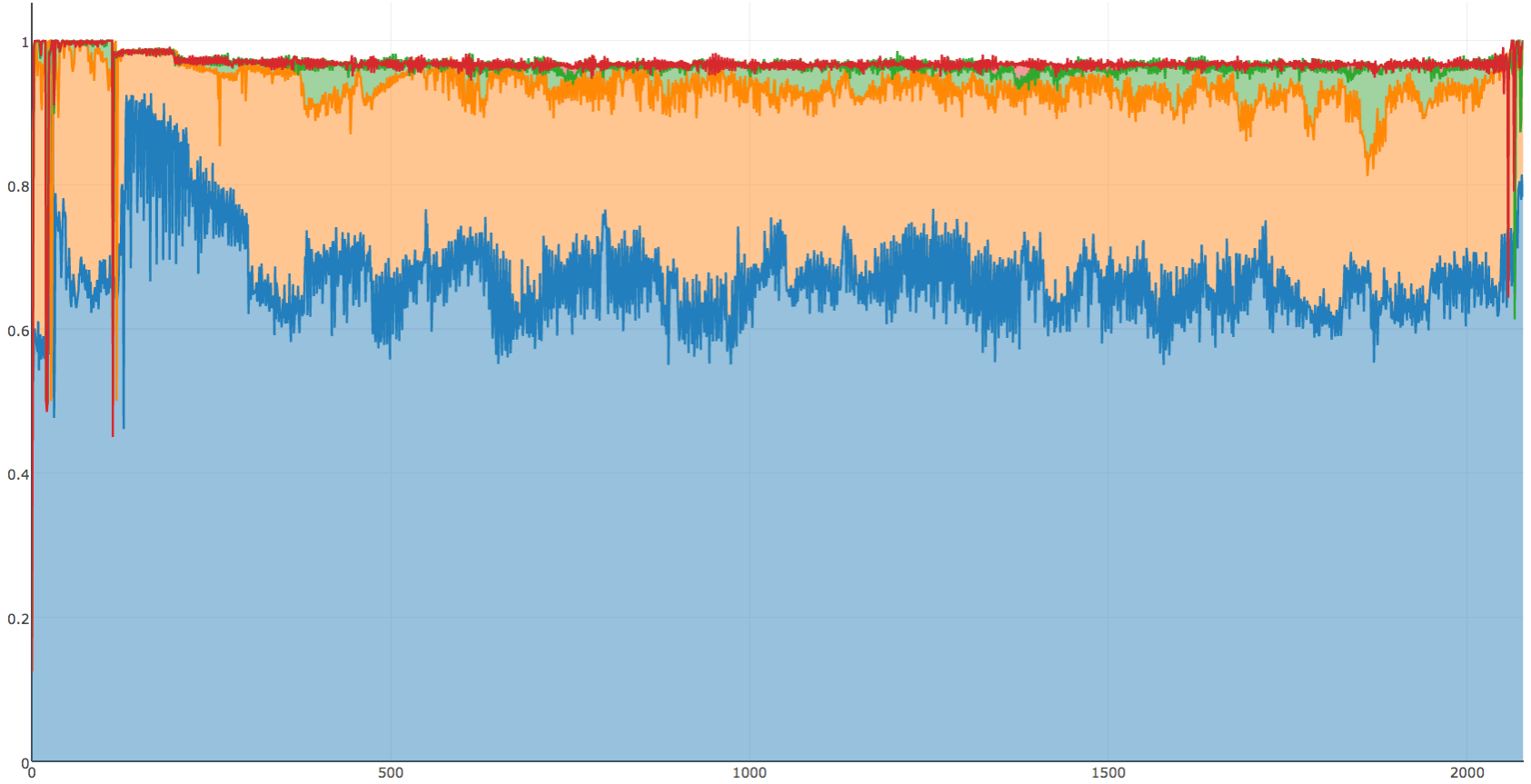
جدول ۲.۵: درصد برخورد حافظه نهان برای آدرس‌های DRAM بارهای کاری بنچمارک رودینیا

Workload/Cache Size	32Kb	48Kb	64Kb	96Kb
BFS	0.407	0.542	0.611	0.648
MRI-GRIDDING	0.484	0.581	0.637	0.665
MRI-Q	0.991	0.993	0.995	0.996
SGEMM	0.368	0.502	0.610	0.639
STENCIL	0.553	0.788	0.835	0.864

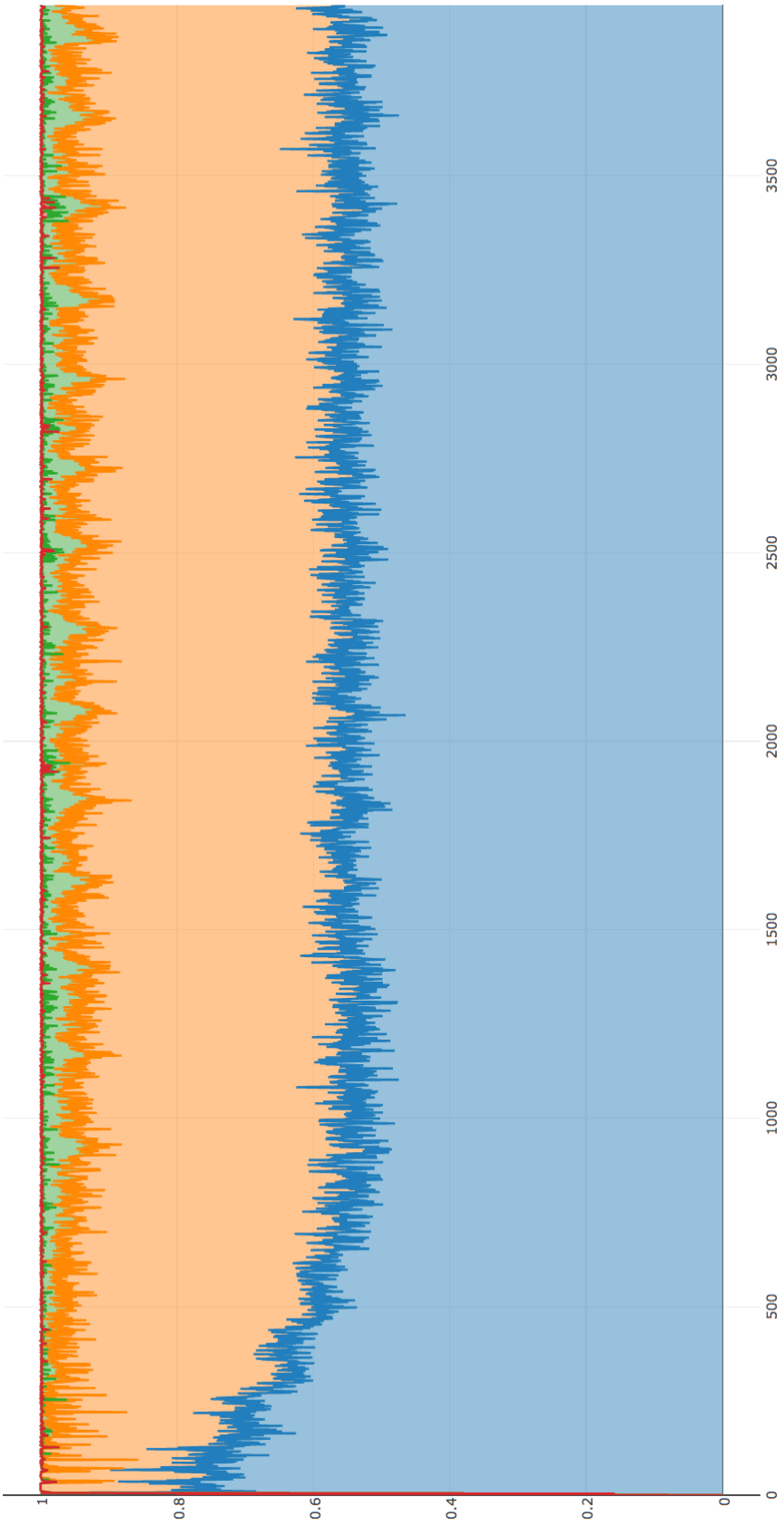
جدول ۳.۵: درصد برخورد حافظه نهان برای آدرس‌های DRAM بارهای کاری بنچمارک پاربویل

با این نتایج مشاهده می‌شود که افزایش حجم حافظه نهان به طور قطع سبب بهبود نرخ برخورد می‌شود. نتایج گزارش شده در جداول فوق میانگین برای تمام پنجره‌های بارکاری مورد نظر بوده است. در حالت کلی مشاهده می‌کنیم که عمده پنجره‌ها در که با افزایش اندازه حافظه نهان نرخ برخورد بالاتری را شاهد هستیم. یک نمونه از هر مجموعه در ادامه آورده شده است.

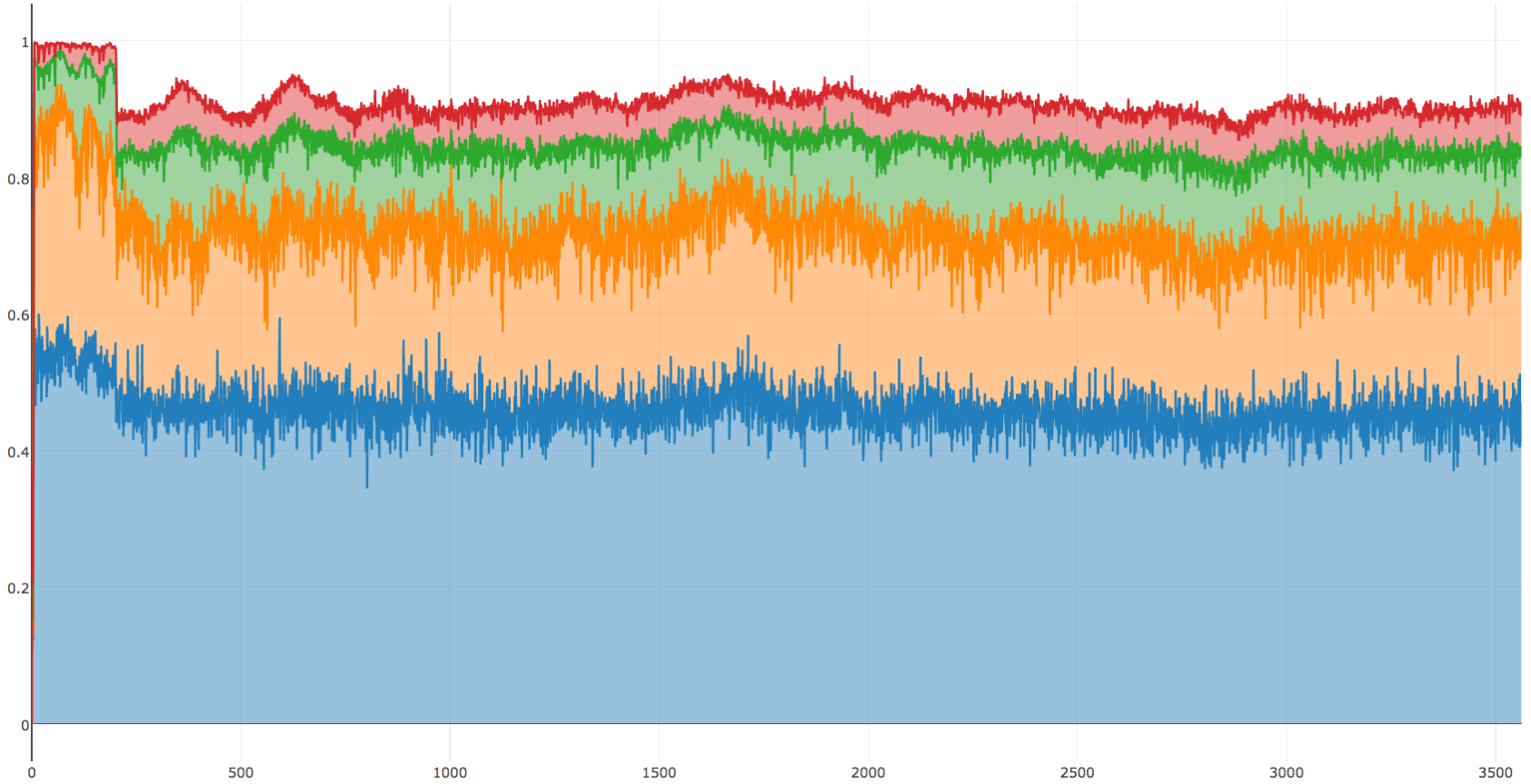
[۲] Page



شکل ۱.۵ : روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری NN از پکیج ISPASS



شکل ۲.۵: روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری SAD از پکیج پاربویل



شکل ۳.۵: روند رشد نرخ برخورد با افزایش اندازه حافظه نهان برای بارکاری Leukocyte از پکیج رودینیا

۳.۵ عملکرد حافظه نهان برای آدرس‌های حافظه مشترک

با توجه به حجم کم حافظه مشترک برای هر چندپردازنده به نظر می‌رسد که نیاز به در نظر گرفتن حافظه نهان چندان بزرگی برای آن نباشد. مشابه قسمت قبل این‌بار فقط آدرس‌های مشترک را به شبیه‌ساز می‌دهیم و نتایج را ثبت می‌کنیم.

Workload/Cache Size	16Kb	32Kb	48Kb	64Kb
LPS	0.999695	0.999691	0.999693	0.999693
NQU	0.986432	0.987453	0.987623	0.988002
STO	0.999987	0.999982	0.999984	0.999991

جدول ۴.۵: درصد برخورد حافظه نهان برای آدرس‌های مشترک بارهای کاری بنچمارک ISPASS

Workload/Cache Size	16Kb	32Kb	48Kb	64Kb
Heart Wall	0.999994	0.999997	0.999996	0.999990
NW	0.991531	0.991709	0.992185	0.992831
Particle Filter	0.784459	0.806113	0.800006	0.827585

جدول ۵.۵: درصد برخورد حافظه نهان برای آدرس‌های مشترک بارهای کاری بنچمارک رودینیا

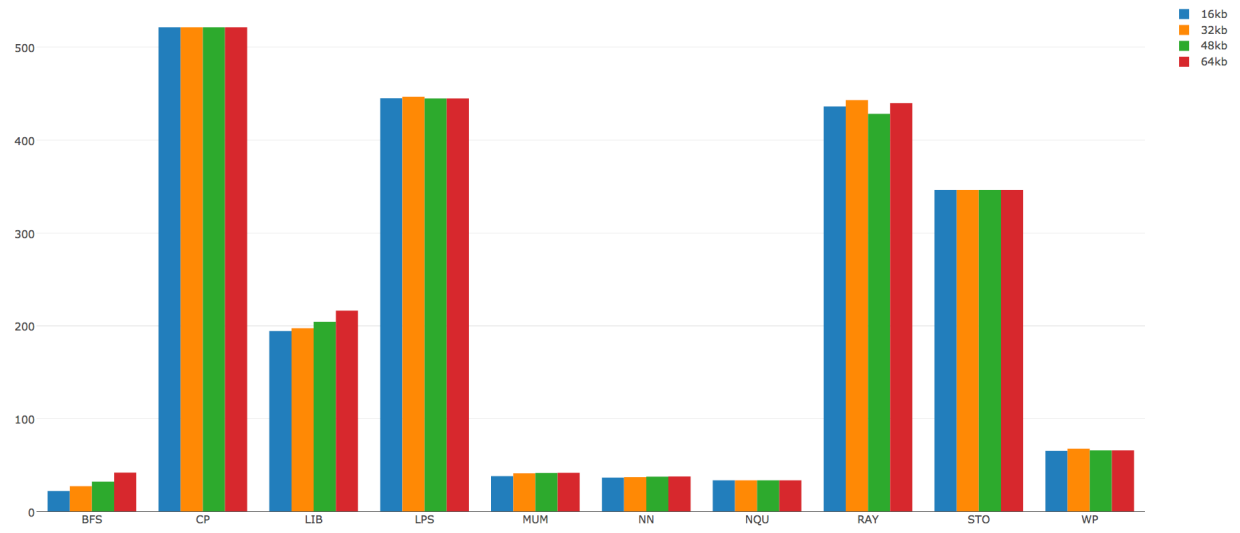
Workload/Cache Size	16Kb	32Kb	48Kb	64Kb
BFS	0.998965	0.999036	0.998903	0.999750
MRI-GRIDDING	0.998132	0.998161	0.998178	0.998232

جدول ۶.۵: درصد برخورد حافظه نهان برای آدرس‌های مشترک بارهای کاری بنچمارک پاربویل

از آن‌جا که تمام اعداد نزدیک به یک هستند داده‌های با احتمال خطای زیاد حذف شده است. در هر صورت با توجه به اندازه کوچک حافظه نهان و نیز پیش بینی الگوهای دسترسی به آن چنین نتایجی مورد انتظار است. همچنین می‌توان نتیجه‌گیری کرد که احتمالا با اندازه‌های حافظه نهان کوچکتر و یا استفاده از یک حافظه نهان نگاشت [۳] مستقیم بتوان نتایج بهتری نیز گرفت. تحلیل دقیق این مورد نیازمند به دست آوردن داده‌های طول عمر آدرس‌ها در حافظه نهان است که جزو قدم‌های بعدی خواهد بود.

۱.۳.۵ بررسی بهبود نرخ دستور در واحد زمان

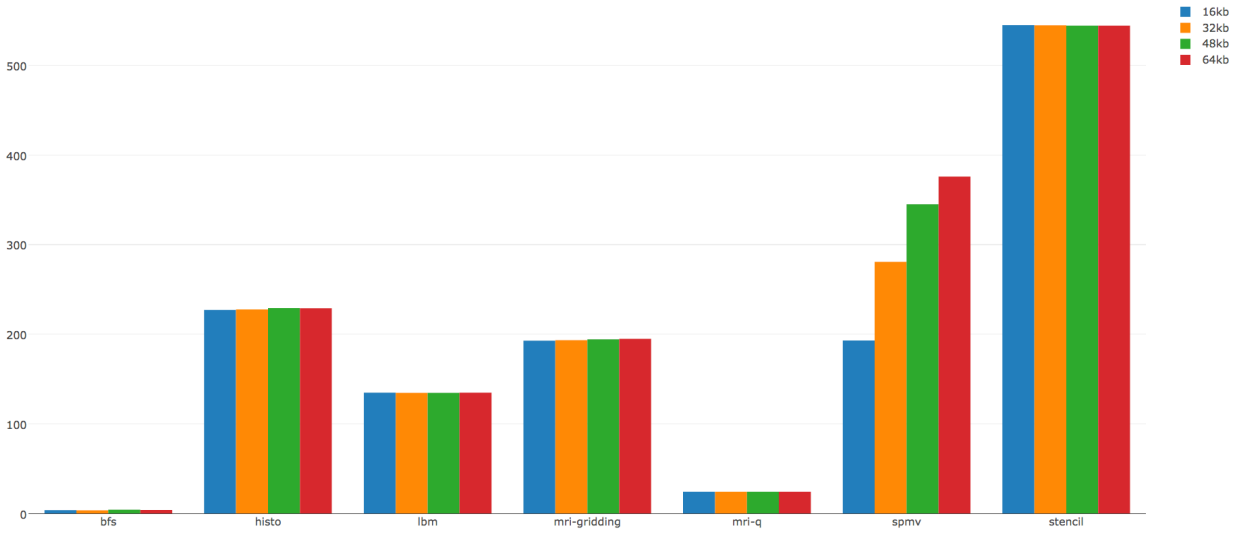
یکی از معیارهای اصلی برای سنجش بهبود عملکرد فاکتور IPC یا تعداد دستورالعمل اجرا شده در واحد زمان است. این کمیت توسط شبیه‌ساز GPGPUSim گزارش داده می‌شود و در نمودارهای زیر میانگین‌آن برای تمام طول اجرای بارهای کاری مختلف را رسم شده است.



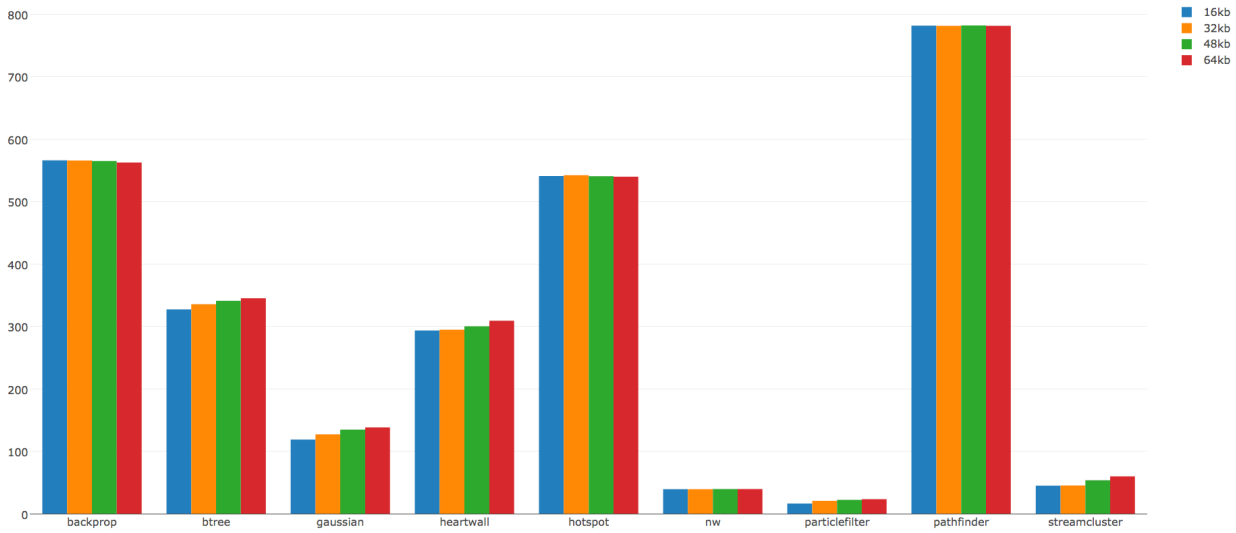
شکل ۴.۵: نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک ISPASS

این نمودارها رشد پیوسته مقادیر IPC با افزایش اندازه حافظه نهان را برای اکثر بارهای کاری نشان می‌دهند. لازم به ذکر است در بعضی موارد ممکن است مقدار IPC به سبب عوامل دیگری مانند وابستگی داده‌ای بین ریسه‌ها و یا پهنای‌باند حافظه پردازنده میزبان محدود شده باشد که در این صورت نباید انتظار بهبودی به واسطه معماری پیشنهادی را داشت.

[۳] Direct Mapped Cache



شکل ۵.۵: نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک پاربویل



شکل ۶.۵: نرخ IPC برای اندازه‌های متفاوت حافظه نهان بنچمارک رودینیا

۶ جمع‌بندی و کارهای آینده

امروزه پردازنده‌های گرافیکی عام‌منظوره به بستری محبوب برای پردازش‌های سریع تبدیل شده‌اند. رابط‌های نرم‌افزاری این پردازنده‌ها به برنامه‌نویسان این امکان را می‌دهند که پردازش‌های خود را در قالب هزاران ریسه که در گروه‌های کوچکتر که هر یک دستور ثابتی را اجرا می‌کنند دسته‌بندی شده‌اند مدل‌سازی کنند. کارهای پیشین نشان داده است که این رویکرد سبب افزایش چندین مرتبه‌ای سرعت برای برخی پردازش‌ها می‌شود.

یکی از عواملی که عملکرد پردازنده‌های گرافیکی را محدود می‌کند استفاده نامناسب [۱] از حافظه مشترک چندپردازنده‌هاست. این امر می‌تواند به علت نیاز به دسترسی ریسه‌ها به داده‌های مشترک از حافظه چرک‌نویس و وابستگی‌داده‌ای و بین آن‌ها و در موارد کمبود پهنای باند این حافظه سبب محدودیت اجرای ریسه‌ها و به تبع آن کاهش *موازات سطح ریسه* [۲] شود. این امر به خصوص در بارهای کاری با محاسبات سنگین می‌تواند کاهش عملکرد پردازنده شود. مطالعات قبلی در این حوزه نشان داده است که پهنای باند حافظه اصلی معمولا بسیار بیشتر از حد مورد نیاز در بارهای پردازشی است. با تکیه بر این دانسته می‌توان امیدوار بود که انتقال ترافیک حافظه مشترک به درگاه DRAM سبب بهبود نرخ TLP شود.

برای استفاده حداکثری از موازات گسترده فراهم شده توسط پردازنده‌های گرافیکی نیاز است تا داده‌های مورد نیاز هر هسته پردازشی در زمان کوتاهی برای آن قابل دسترسی باشد و عملکرد مناسب حافظه نهان می‌تواند در اینجا بسیار حیاتی باشد. تا به اینجای کار نشان داده‌ایم که افزایش حجم حافظه نهان به یقین تاثیر مثبت قابل توجهی رو نرخ برخورد و نیز مقادیر IPC برای بارهای کاری عام‌منظوره داشته باشد. همچنین می‌بینیم که اختصاص قسمت کوچکی از حافظه نهان هر چندپردازنده به فضای آدرس مشترک می‌تواند تضمین‌کننده نرخ برخورد بالا باشد. این مشاهدات شهود اولیه ما را تایید می‌کند ولی همچنان نیاز به بررسی دقیق‌تر برای امکان‌سنجی پیاده‌سازی این معماری وجود دارد.

[۱] Underutilization

[۲] Thread Level Parallelism (TLP)

در نظر داشته باشیم که تا به اینجای کار هنوز تاثیر تاخیر ایجاد شده به واسطه انتقال داده‌های فضای مشترک به حافظه اصلی را در نظر گرفته نشده است. همچنین تاثیر بار اضافه شده بر درگاه حافظه DRAM بر عملکرد سایر قسمت‌های پردازنده نیز باید به نوعی در نتیجه‌گیری لحاظ شود. از مواردی که می‌تواند در ادامه‌ی این پروژه قرار بگیرد اعمال تغییرات لازم در شبیه‌ساز به منظور نگاشت فضای آدرس‌دهی مشترک به حافظه اصلی است. همچنین با تحلیل دقیق‌تر الگوی دسترسی به حافظه‌نهان و حافظه مشترک و نیز اضافه کردن بارهای کاری جدید به مجموعه فعلی می‌توان یک الگوریتم واکشی‌پیش‌دستانه Pre-Fetching طراحی کرد تا عملکرد حافظه نهان را به خصوص در هنگام بازگرداندن داده‌های قدیمی به حافظه اصلی بهبود بخشید.

کتاب‌نامه

[۱] S. Borkar, *"Design challenges of technology scaling"* in IEEE Micro, vol. 19, no. 4, pp. 23-29, Jul-Aug 1999.

[۲] M. D. Hill and M. R. Marty, *"Amdahl's Law in the Multicore Era,"* in Computer, vol. 41, no. 7, pp. 33-38, July 2008.

[۳] Nvidia Corportaion, *" Nvidia Kepler GK110 Architecture Whitepaper*"

[۴] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous and A. R. LeBlanc, *"Design of ion-implanted MOSFET's with very small physical dimensions,"* in IEEE Journal of Solid-State Circuits, vol. 9, no. 5, pp. 256-268, Oct 1974.

[۵] D. Luebke, *"CUDA: Scalable parallel programming for high-performance scientific computing"*, 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Paris, 2008, pp. 836-838.

[۶] Sorensen, Ganesh Gopalakrishnan, and Vinod Grover. 2013. *"Towards shared memory consistency models for GPUs"*. In Proceedings of the 27th international ACM conference on International conference on supercomputing (ICS '13)

[۷] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, *"Analyzing CUDA workloads using a detailed GPU simulator," 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston"*, MA, 2009, pp. 163-174.

[۸] S. Che et al., *"Rodinia: A benchmark suite for heterogeneous computing,"* 2009 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, 2009, pp. 44-54.

[۹] Che, Shuai, Jeremy W. Sheaffer, Michael Boyer, Lukasz G. Szafaryn, Liang Wang, and Kevin Skadron. *"A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads."* In Workload Characterization (IISWC), 2010 IEEE International Symposium on, pp. 1-11. IEEE, 2010.

[۱۰] Stratton, John A., et al. *"Parboil: A revised benchmark suite for scientific and commercial throughput computing."* Center for Reliable and High-Performance Computing 127 (2012).

[۱۱] Kayıran, O., Jog, A., Kandemir, M.T. and Das, C.R., 2013, October. *"Neither more nor less: optimizing thread-level parallelism for GPGPUs"*. In Proceedings of the 22nd international conference on Parallel architectures and compilation techniques (pp. 157-166). IEEE Press.

[۱۲] Jatala, Vishwesh, Jayvant Anantpur, and Amey Karkare. *"Scratchpad sharing in GPUs."* arXiv preprint arXiv:1607.03238 (2016).

[۱۳] van den Braak, G.J., Gómez-Luna, J., Corporaal, H., Gonzalez-Linares, J.M. and Guil, N., 2013, October. *"Simulation and architecture improvements of atomic operations on GPU scratchpad memory"*. In Computer Design (ICCD), 2013 IEEE 31st International Conference on (pp. 357-362). IEEE.

[۱۴] Puzak, Thomas Roberts. *"Analysis of cache replacement-algorithms."* (1985).

[۱۵] Gao, Shuang, and Gregory D. Peterson. *"Optimizing CUDA Shared Memory Usage."*

@pmargs

@title

Abstract

Sharif University of Technology
Department of Computer Engineering

B.Sc Thesis
Software Engineering

A Study on Performance of Shared Memory in GPGPUs

By
Parsoa Khorsand RahimZadeh

Supervisor
Dr. Hamid Sarbazi Azad

June 25, 2017