دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

بهبود کارایی پردازنده‌های گرافیکی با افزایش بهره‌برداری از SIMD

نگارش

احمد لشگر

استاد راهنمای اول: **دکتر احمد خونساری**

استاد راهنمای دوم: **دکتر امیرعلی بنی‌اسدی**

پایان‌نامه برای دریافت درجه کارشناسی ارشد

در

رشته مهندسی کامپیوتر-معماری کامپیوتر

آبان ۱۳۹۱

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

پایان نامه برای دریافت درجه کارشناسی ارشد در رشته مهندسی کامپیوتر – معماری کامپیوتر

عنوان:

بهبود کارایی پردازنده‌های گرافیکی با افزایش بهره‌برداری از SIMD

نگارش: احمد لشگر

این پایان‌نامه در تاریخ ۱۳۹۱/۰۸/۲۲ در مقابل هیات داوران دفاع گردید و مورد تصویب قرار گرفت.

معاون آموزشی و تحصیلات تکمیلی پردیس دانشکده‌های فنی: دکتر علی افضلی کوشا

رئیس دانشکده برق و کامپیوتر: دکتر جلیل آقا راشد محصل

معاون پژوهشی و تحصیلات تکمیلی دانشکده مهندسی برق و کامپیوتر: دکتر سعید افشارنیا

استاد راهنمای اول: دکتر احمد خونساری	امضاء:
استاد راهنمای دوم: دکتر امیرعلی بنی‌اسدی	امضاء:
عضو هیات داوران: دکتر سعید صفری	امضاء:
عضو هیات داوران: دکتر حمید سربازی آزاد	امضاء:
عضو هیات داوران: دکتر سیدمهدی فخرایی	امضاء:

تعهد اصالت اثر

اینجانب احمد لشگر تایید می‌نمایم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب است و به دستاوردهای پژوهشی دیگران که در این نوشته از آنها استفاده شده است طبق مقررات ارجاع گردیده است. این پایان نامه قبلا برای احراز هیچ مدرک هم سطح یا بالاتر ارائه نشده است.

نام و نام خانوادگی: احمد لشگر

امضاء:

تقدیم به روح پدربزرگ هایم

که نایل شدن به تحصیلات عالی را

برای نوه‌های خود آرزو داشتند.

تشکر

از خانواده‌ی عزیزم، پدر، مادر، و برادرم که باارزش‌ترینِ داشته‌هایم هستند، سپاس‌گزارم که در طول این تحقیق همواره مشوق و دلگرمی دهنده‌ی اینجانب بودند و با مهیا کردن محیطی عاری از دغدغه، آرامش خاطر کامل برای انجام پژوهش را فراهم آوردند.

از دکتر احمد خونساری، استاد راهنمای اول این پژوهش، که همواره با حداکثر اشتیاق انگیزه‌بخش این پژوهش بودند، به پاس یاری همه‌جانبه و مهیا کردن امکانات مورد نیاز این پژوهش و انتقال دانش‌هایی از جمله مدیریت زمان و هوش اجتماعی، سپاس فراوان دارم. از دکتر امیرعلی بنی‌اسدی، استاد راهنمای دوم این پژوهش، که با هدایت خردمندانه تحقیق در به ثمر رساندن آن نقش کلیدی داشتند، و با حداکثر انرژی دانش‌هایی از قبیل نگارش متن علمی، انجام تحقیق، و ارتباط اجتماعی را به اینجانب آموختند، صمیمانه تقدیر می‌کنم.

از اساتید گرامی، دکتر سید مهدی فخرایی، دکتر حمید سربازی‌آزاد، و دکتر سعید صفری، به پاس قبول داوری این پژوهش، و ارائه نظرات گران‌بها در جهت بهبود کیفیت آن سپاس فراوان دارم.

تشکر ویژه دارم از دکتر امین‌الله مهابادی به پاس زحمات فراوانی که برای بهبود کیفیت نگارش این پایان‌نامه متقبل شدند. همچنین قدردان هستم از دوستان گرامی، علی شفیعی و بهنام خدابنده‌لو که در راستای بهبود کیفیت تکنیک‌های ارائه شده در این تحقیق نظرات ارزشمندی ارائه کردند.

چکیده

پردازنده‌های گرافیکی به عنوان شتاب‌دهنده در بسیاری از ابررایانه‌های جهان مورد استفاده قرار می‌گیرند. در این پردازنده‌ها هزاران نخ همروند در قالب گروه‌هایی در دسته‌های چنده‌تایی زمان‌بندی می‌شوند و یک دستور مشترک را روی محاسبه‌گرهای SIMD اجرا می‌کنند. زمانی که دستور مشترک گروه، انشعاب شرطی باشد، نخ‌ها به مسیرهای مختلفی واگرا می‌شوند. روش مرسوم برای مدیریت این پدیده در پردازنده‌های گرافیکی واحدهای محاسباتی را به میزان چشمگیری بیکار می‌گذارد. در این پژوهش، این پدیده را عمیقا بررسی می‌کنیم و پی‌آمدهای آن را تحلیل می‌کنیم. همچنین نشان می‌دهیم که اندازه گروه‌بندی نخ‌ها می‌تواند بر گذردهی این پردازنده‌ها موثر باشد. مبتنی بر مشاهدات فوق، دو تکنیک معماری برای بهبود گذردهی در این پردازنده‌ها ارائه می‌کنیم. روش اول CROWN نام دارد و تکنیکی برای گروه‌بندی پویای نخ‌های واگرا شده و دوباره همگرا کردن آنها است. ارزیابی‌ها نشان می‌دهد که این تکنیک با سربار تخمینی سخت‌افزاری ۴٪، بروندهی را تا ۲.۳ برابر بهبود می‌دهد. روش دوم DWR نام دارد که تکنیکی برای نایل شدن به مزایای هر دو اندازه گروه‌بندی بزرگ و کوچک ارائه می‌کند. ارزیابی‌های ما نشان می‌دهد که این تکنیک با سربار سخت‌افزاری کمتر از ۱٪، قادر است بروندهی را تا ۷۰٪ بهبود بخشد.

واژه‌های کلیدی: معماری پردازنده، شتاب‌دهنده سخت‌افزاری، پردازنده گرافیکی، بهره‌برداری از SIMD، چند نخی، گروه‌بندی نخ‌ها.

فهرست مطالب

فهرست شکل‌ها

فهرست جداول

فهرست الگوریتم‌ها

مخفف‌ها

CPU……. Central Processing Unit
CROWN…….. Comprehensive ReincarnatiOn-based WarpiNg
CUDA……. Compute Uniform Device Architecture
DRAM……. Dynamic Random Access Memory
DWF…… Dynamic Warp Formation
DWR……... Dynamic Warp Resizing
GPGPU……... General-Purpos GPU
GPU……. Graphics Processing Unit
ID…… Identifier
ILT…..……. Ignore-List Table
ISA……... Instruction Set Architecture
LWM……... Large-Warp Micro-architecture
MIMD…… Multiple-Instruction Multiple-Data
PC……..... Program Counter
PST…..……. Partner-Synch Table
PTX……..Parallel Thread Execution
RISC…….Reduced Instruction Set Computing
RPC…..…….Re-convergence Program Counter
SBR……. Stack-Based Re-convergence
SCO……..Sub-warp Combiner
SIMD……... Single-Instruction Multiple-Data
SIMT……... Single-Instruction Multiple-Thread
SM……... Streaming Multiprocessor
SPMD…….. Single-Program Multiple-Data
TBC……..... Thread-Block Compaction
TID…….. Thread Identifier
WID……. Warp Identifier

فصل ۱

کلیّات

۱-۱ مقدمه

در سال‌های اخیر گونه‌ی جدید از پردازشگرهای موازی با مدل پردازشی SIMT[1] مطرح شده‌اند [۳۴]. مدل SIMT، از مصالحه ویژگی‌های کلیدی مدل‌های مرسوم MIMD و SIMD ایجاد شده است. در لایه‌ی نرم‌افزارِ SIMT، میلیون‌ها نخ با مدل برنامه‌نویسی MIMD برنامه‌ریزی می‌شود و در لایه سخت‌افزار، نخ‌ها گروه‌بندی شده و روی ساختار SIMD اجرا می‌شوند. در نتیجه هدف SIMT را می‌توان کاهش انرژی مصرفی با حفظ سهولت برنامه‌نویسی دانست. پردازش همه‌منظوره[2] روی پردازنده‌های گرافیکی[3] (GPU) نمونه‌ای از تحقق مدل SIMT است [۳۴، ۴۷، ۴۰]. این پردازنده‌ها برای گذردهی[4] طراحی شده‌اند و عموما تاخیر اجرای یک دستور در آنها زیاد است [۴۸]. در پردازش‌های با کارایی بالا[5]، GPUها به عنوان شتاب‌دهنده در کنار پردازنده مرکزی در یک سیستم ناهمگون[6] استفاده می‌شوند. در این ساختار، برنامه‌های سریال (یا با درجه‌ی کمی از موازی‌سازی) از پردازنده مرکزی استفاده می‌کنند و برنامه‌هایی با درجه موازی‌سازی عظیم از شتاب‌دهنده استفاده می‌کنند.

معماری پردازنده‌های SIMT از شبکه‌برتراشه‌ای از محاسبه‌گرها و کنترل‌کننده‌های حافظه تشکیل شده است. هر محاسبه‌گر محتوای هزاران نخ را نگهداری می‌کند و گذردهی را با همپوشانی دادن زمان دسترسی به حافظه‌ی دسته‌ای از نخ‌ها با محاسبات دسته‌ای دیگر، بالا نگه می‌دارد. برای نایل شدن به طراحی کارآمد (در هر دو جنبه گذردهی و انرژی) در زمان‌بندی نخ‌ها، هر محاسبه‌گر دسته‌ای از نخ‌ها را در قالب واحدی

[1] Single Instruction Multiple Threads
[2] General-purpose computing
[3] Graphics processing unit
[4] Throughput
[5] High performance computing
[6] Heterogeneous

بزرگتر به نام وارپ[7] دسته‌بندی می‌کند و همگام پیش می‌برد و نخ‌های فعال یک وارپ همواره یک دستور را اجرا می‌کنند. با زمان‌بندی در دانه‌بندی وارپ، چندین مرتبه از پیچیدگی زمان‌بندی نخ‌ها کاسته می‌شود [۱۶]. افزون بر این، استفاده از وارپ فرصتی مهیا می‌کند تا دسترسی به حافظه‌ی چندین نخ همزمان اجرا شود و از محلیت دسترسی‌ها برای کاهش تعداد درخواست‌های حافظه بهره برده شود. به این ویژگی الحاق دسترسی‌ها به حافظه[8] می‌گویند که نقش مهمی در کاهش تعداد درخواست‌های حافظه (و تبعا افزایش بروندهی) دارد. در معماری‌های مختلف، الحاق دسترسی‌ها به حافظه روی کل وارپ یا بخشی از وارپ انجام شود.

با بهره‌بردن از نظم موجود در بارکاری‌ها، پردازنده‌های SIMT گاهی سریع‌تر (از لحاظ زمان اجرا) و کم‌مصرف‌تر (از لحاظ مصرف توان) از پردازنده‌های MIMD ظاهر می‌شوند [۳۱]. از سوی دیگر، نسبت گذردهی به انرژی مصرفی در پردازنده‌های SIMT بسیار بزرگتر از پردازنده‌های چندهسته‌ای متقارن متداول است [۲۲]. بدین دلیل، شتاب‌دهنده‌های SIMT در بسیاری از ابررایانه‌های جهان، که نسبت گذردهی به مصرف انرژی فاکتوری کلیدی در طراحی است، بکار گرفته می‌شوند (از جمله Titan، Tianhe-1A، و Nebulae) [۴۶]. علی رغم پیشرفت های چشم‌گیر اخیر در معماری GPUها، فاصله‌ی بزرگی بین کارایی بیشینه و کارایی LINPACK [۴۳] اندازه‌گیری شده در این ابررایانه‌ها وجود دارد. همان گونه که توسط Top500.org گزارش شده است، Titan، Tianhe-1A، و Nebulae به ترتیب تنها به ۶۵٪، ۵۵٪، و ۴۳٪ از کارایی بیشینه‌ی خود رسیده‌اند. بیکاری منابع محاسباتی یکی از دلایل اصلی پشت این فاصله‌ی بزرگ است.

دو واگرایی مهم بر بیکاری منابع در سریع‌کننده‌های SIMT تاثیرگذار هستند: واگرایی انشعاب و واگرایی حافظه [۳۵]. واگرایی انشعاب وقتی رخ می‌دهد که وارپ دستورِ انشعابِ شرطی را اجرا کند و شرطِ دستورِ کنترلی برای نخ‌های یک وارپ ناهمسان سنجیده شود. این رخ‌داد باعث می‌شود که نخ‌های وارپ به مسیرهای اجرای دستورِ متفاوتی واگرا شوند. روش‌های مرسوم مدیریت این رخ‌داد در GPUهای امروزی، واحدهای محاسباتی را به مقدار قابل ملاحظه‌ای بیکار می‌گذارد [۱۳]. واگرایی حافظه، در هنگام دسترسی نامنظم نخ‌های همسایه به حافظه زمانی رخ می‌دهد که دسته‌ای از نخ‌ها داده خود را در حافظه نهان می‌یابند

[7] Warp
[8] Memory access coalescing

و دسته‌ای دیگر با فقدان روبرو می‌شوند و باید به حافظه سراسری دسترسی پیدا کنند. این واگرایی باعث توقف غیرالزامی نخ‌هایی می‌شود که داده خود را در حافظه نهان پیدا کرده‌اند.

برای نایل شدن به نهایت بروندهی این پردازنده‌ها، جلوی خطلوله[9] مسئول است بهره‌وری بالایی از ۱) عمق خطلوله و ۲) پهنای SIMD عقب خطلوله[10] فراهم کند. واگرایی انشعاب بهره‌وری از پهنای SIMD خطلوله را کاهش می‌دهد و واگرایی حافظه وارپ‌های فعال را کم می‌کند که کاهش بهره‌وری از عمق خطلوله را در پی دارد.

۱-۲ نوآوری‌ها

در این پژوهش دو مشاهده‌ی کلیدی صورت گرفته است. در مشاهده اول، با تحلیل واگرایی انشعاب نشان می‌دهیم که بهبود نرخ فعالیت SIMD تنها یکی از سه راس مثلثی است که برای مقابله با افت گذردهی ناشی از واگرایی انشعاب هدف قرار می‌گیرد و در برخی موارد دو عامل دیگر بیش از نرخ فعالیت SIMD اهمیت پیدا می‌کنند. سه راس این مثلث را به تفصیل معرفی می‌کنیم و میزان اثرگذاری هریک را به‌صورت عددی گزارش می‌کنیم و نشان می‌دهیم در چه مواردی دو عامل دیگر بیش از نرخ فعالیت SIMD اهمیت پیدا می‌کنند. در مشاهده دوم، نشان می‌دهیم زمانی که الحاق دسترسی‌ها به حافظه روی کل وارپ انجام می‌شود، اندازه وارپ (تعداد نخ‌های هر وارپ) پارامتری تاثیرگذار بر گذردهی پردازنده است. **وارپ‌های کوچک**، به اندازه تعداد خطوط SIMD، رخ‌داد واگرایی انشعاب و حافظه را کم می‌کنند. کاهش واگرایی انشعاب ۱) تعداد خطوط غیرفعال در مسیرهای واگرایی و ۲) نخ‌های منتظر بر نقطه همگرایی را کم می‌کند و کاهش واگرایی حافظه، نخ‌هایی را که داده خود را در حافظه نهان پیدا می‌کنند کمتر منتظر نگه می‌دارد. در طرف منفی، وارپ‌های کوچک الحاق دسترسی به حافظه را کمتر می‌کنند که خود باعث دسترسی‌های تکراری به حافظه اصلی می‌شود. این اتفاق باعث می‌شود تاخیر دسترسی به حافظه اصلی افزایش پیدا کند. **وارپ‌های بزرگ**، از طرف دیگر، از محلیت دسترسی‌های حافظه می‌توانند بهره ببرند و تعداد دسترسی‌های خارج از هسته را کم کنند. در طرف منفی، وارپ‌های بزرگ واگرایی انشعاب و حافظه را تشدید می‌کنند.

[9] Pipeline Frontend
[10] Pipeline Backend

مبتنی بر مشاهدات انجام شده در این پژوهش، برای بهبود گذردهی در پردازنده‌های SIMT دو راه حل پیشنهاد می‌کنیم. برای معماری پایه‌ای که الحاق دسترسی‌ها به حافظه را روی بخشی از وارپ (به پهنای SIMD) انجام می‌دهد، روش جامع باززنده‌شدن در اجرای وارپی[11] (CROWN) را ارائه می‌کنیم. این روش برای مقابله جامع با سه راس مثلث تاثیرگذار در واگرایی انشعاب طراحی شده است و با گروه‌بندی پویای نخ‌ها در هنگام واگرایی و با باززنده کردن وارپ‌های واگرا شده در این جهت حرکت می‌کند. در روش پیشنهادی دوم، برای معماری پایه‌ای که الحاق دسترسی‌ها روی کل وارپ را انجام می‌دهد، روش تغییر اندازه وارپ پویا[12] (DWR) را پیشنهاد می‌کنیم [۲۸] تا به مزایای هر دو طراحی وارپ‌های بزرگ و کوچک دست پیدا کنیم.

برای پیاده‌سازی هریک از روش‌های پیشنهادی، معماری واقعی ارائه می‌کنیم و هزینه سخت‌افزاری این پیاده‌سازی را ارزیابی می‌کنیم. افزون بر این، یافته‌های خود را در هر روش روی طیف گسترده‌ای از پیکربندی‌ها باز ارزیابی می‌کنیم تا کارایی روش‌های پیشنهادی برای معماری‌های متفاوتی سنجیده شود.

۱-۳ ساختار پایان‌نامه

ساختار ادامه‌ی این پایان‌نامه از قرار زیر است. در فصل ۲ به مرور زمینه پژوهش و مشاهدات انگیزه‌بخش آن می‌پردازیم. در فصل ۳ کارهای انجام شده در زمینه مرتبط با این پژوهش را مرور می‌کنیم. در فصل ۴ روش پیشنهادی CROWN را معرفی می‌کنیم. در فصل ۵ روش پیشنهادی DWR را معرفی می‌کنیم. در فصل ۶ تنظیمات و محیط شبیه‌سازی معرفی می‌شود و نتایج شبیه‌سازی برای ارزیابی CROWN و DWR را ارائه می‌کنیم. در فصل ۷ نتیجه‌گیری می‌کنیم و برخی از کارهای آینده را که به آنها خواهیم پرداخت مطرح می‌کنیم.

[11] Comprehensive Reincarnation-based Warping
[12] Dynamic Warp Resizing

فصل ۲

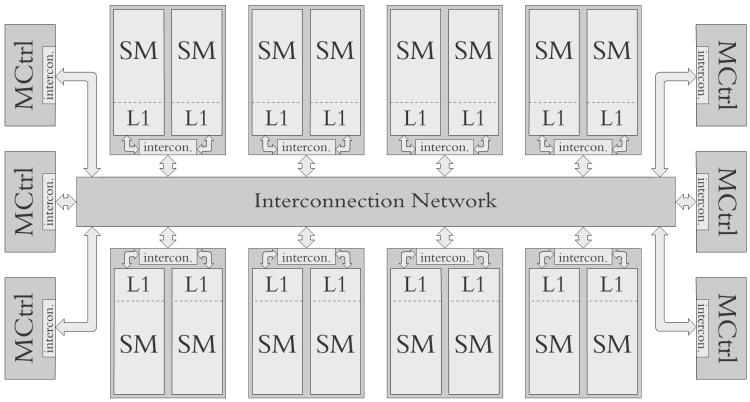
پیش‌زمینه

در این فصل، پیش‌زمینه‌ای از مباحث مرتبط با این پژوهش مطرح می‌کنیم. در ابتدا مدل پردازشی GPUها در محاسبات همه‌منظوره را شرح می‌دهیم. سپس معماری GPU و پردازش وارپی را معرفی می‌کنیم و بعد الحاق دسترسی به حافظه را شرح می‌دهیم. در ادامه دو نوع واگرایی انشعاب[۱] و واگرایی حافظه[۲] را معرفی می‌کنیم و طریقه مدیریت آنها را شرح می‌دهیم. در پایان مشاهدات انگیزه‌بخش این پژوهش درباره‌ی واگرایی انشعاب و تاثیر اندازه وارپ را ارائه می‌کنیم.

۲-۱ مدل پردازشی **GPU**ها

در مدل‌هایی مانند کودا[۳] [۳۹] که از GPU برای پردازش‌های همه‌منظوره استفاده می‌شود، GPU پردازنده‌ای کمکی برای CPU به‌حساب می‌آیند. این پردازنده کمکی می‌تواند روی یک تراشه در کنار CPU باشد یا از طریق کارت جانبی (مانند PCI-E) به سیستم اضافه شود. در این مدل، در ابتدا پروسس اجراشده روی CPU (که پروسس میزبان نامیده می‌شود)، بارکاری برای GPU می‌فرستد و خود پردازش‌های غیروابسته دیگری را موازی با GPU ادامه می‌دهد. بارکاری از جنس پردازش SPMD[۴] است که در آن تعداد معین‌شده‌ای نخ موازی، تابعی به نام کرنل[۵] را اجرا می‌کنند. در بدنه‌ی کرنل، نخ‌ها از طریق شناسه‌ای منحصر بفرد، خود را از یکدیگر متمایز می‌کنند و به داده‌های متفاوتی دسترسی پیدا می‌کنند. نخ میزبان اغلب منتظر اتمام اجرای بارکاری می‌شود تا به داده‌ی پردازش شده‌ی معتبر دسترسی پیدا کند.

[1] Branch divergence
[2] Memory divergence
[3] Compute Unified Device Architecture (CUDA)
[4] Single Program Multiple Data
[5] Kernel



شکل ۲-۱: معماری GPU مطالعه شده در این پژوهش. کنترل کننده‌های حافظه با MCtrl (Memory Controller) و هسته‌ها با SM (Streaming Multiprocessor) نشان داده شده‌اند.

مجموعه‌ی نخ‌هایی که کرنل را اجرا می‌کنند، گرید[6] نام دارند. تعداد نخ‌های گرید می‌تواند بسیار زیاد باشد (حدود $۲^{۳۱}-۱$ نخ) ولی هر هسته تنها هزاران نخ را می‌تواند همزمان اجرا کند. برای آنکه اجرا تمام نخ‌های گرید با محدودیت‌های سخت‌افزاری ممکن باشد، گرید توسط برنامه‌نویس به چندین بلوک نخ[7] تقسیم می‌شود. هر هسته‌ی GPU محتوای نخ‌های تعدادی بلوک (که بلوک‌های فعال نام دارند) را در خود نگه می‌دارد و باقی بلوک‌ها، منتظر اتمام اجرای بلوک‌های فعال می‌مانند. پس از اتمام اجرای بلوک‌های فعال، باقی بلوک‌ها اجرا می‌شوند تا تمامی بلوک‌های گرید کامل شوند. نخ‌های یک بلوک روی یک هسته اجرا می‌شوند و امکان تبادل اطلاعات از طریق حافظه‌ای سریع روی تراشه (از جنس حافظه نهان) را دارند و برای جلوگیری از شرایط رقابت[8]، امکان همگام کردن آنها (نخ‌های یک بلوک) وجود دارد. گروه‌بندی نخ‌ها در گرید امکان بهره‌بردن از اصل محلیت داده و همچنین کاهش سربار تبادل داده بین نخ‌ها را فراهم می‌کند.

۲-۲ معماری GPU

۲-۲-۱ سطح بالا

در این پژوهش، GPUای شبیه به معماری NVIDIA Tesla [۳۴] مدل شده است که متشکل از شبکه‌ای برتراشه از هسته‌ها و کنترل‌کننده‌های حافظه است. این ساختار در شکل ۲-۱ نشان داده شده است. هسته‌ها

[6] Grid
[7] Thread block
[8] Race conditions

درخواستِ حافظه‌ی نخ‌ها را از طریق شبکه‌ی برتراشه به کنترل‌کننده حافظه[9] مرتبط می‌فرستند و شبکه برتراشه داده‌ی خواسته‌شده (یا تایید نوشتن) را از کنترل‌کننده حافظه به هسته باز می‌رساند.

هر هسته چندین بلوک نخ را اجرا می‌کند. بلوک‌های نخ توسط هسته به چند وارپ تقسیم می‌شوند. وارپ، گروهی از نخ‌ها (عموما ۳۲ نخ) است که همگام با هم پیش می‌روند و روی یک SIMD اجرا می‌شوند. نخ‌های یک وارپ همواره پس از اجرای هر دستور همگام می‌شوند. وارپ‌های یک بلوک مستقل از یکدیگر زمان‌بندی شده [۳۳] و از طریق دستور ویژه‌ای با هم ارتباط برقرار می‌کنند و همگام می‌شوند.

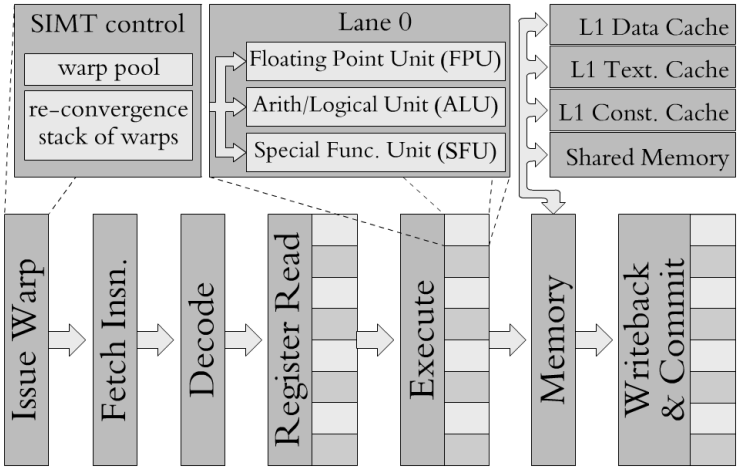
۲-۲-۲ فضای حافظه‌ای

۶ دسته فضای حافظه‌ای متداول در GPU ها وجود دارد: ۱) حافظه ثباتی، ۲) حافظه محلی، ۳) حافظه مشترک[10]، ۴) حافظه داده، ۵) حافظه ثابت، و ۶) حافظه بافت[11] [۳۹]. حافظه ثباتی و حافظه محلی حاوی متغیرهای خصوصی هر نخ هستند و تنها خود نخ به آنها دسترسی دارد. حافظه مشترک برای هر بلوک تعریف شده است و نخ‌های هر بلوک تنها می‌توانند به حافظه مشترکِ بلوکِ خود دسترسی پیدا کنند. حافظه داده، ثابت، و بافت بین تمامی نخ‌های گرید قابل رویت هستند و همگی به‌آن دسترسی دارند. حافظه داده، ثابت، و بافت حافظه سراسری[12] نیز نام دارند. در ادامه این فضاهای حافظه‌ای را معرفی می‌کنیم.

حافظه ثباتی توسط کامپایلر مدیریت می‌شود و سریع‌ترین نوع حافظه است که در رجیسترفایل[13] هر هسته نگهداری می‌شود. حافظه محلی توسط کامپایلر مدیریت می‌شود و برای نگهداری ثبات‌هایی که به علت محدودیت حجم رجیسترفایل باید خارج از آن نگهداری شوند، بکار می‌رود. حافظه محلی عموما در DRAM خارج از تراشه ذخیره می‌شود. حافظه ثباتی و محلی خواندنی و نوشتی هستند.

حافظه مشترک خواندنی و نوشتنی است و در هسته نگهداری می‌شود. حافظه مشترک کندتر از رجیسترفایل و سریع‌تر از DRAM خارج تراشه است. داده‌های حافظه‌های مشترک در محل دیگری ذخیره نشده‌اند و با آغاز اجرای بلوک تخصیص و با پایان اجرای بلوک آزاد می‌شوند. مدیریت این حافظه

[9] Memory Controller
[10] Shared memory
[11] Texture memory
[12] Global memory
[13] Register file



شکل ۲-۲: ساختار خطلوله هر هسته.

نرم‌افزاری است و توسط برنامه‌نویس انجام می‌شود. حافظه مشترک اغلب برای محلی کردن و تسریع ارتباطات بین نخ‌های یک بلوک بکار می‌رود.

حافظه داده، ثابت، و بافت روی DRAM خارج از تراشه‌ی GPU ذخیره می‌شود و هریک روی سلسله مراتب جداگانه‌ای در تراشه، نهان[۱۴] می‌شوند. حافظه داده متداول‌ترین نوع حافظه است که برای خواندن و نوشتن استفاده می‌شود. حافظه ثابت مشابه حافظه داده است با این تفاوت که حافظه‌ای فقط خواندنی هستند. حافظه بافت تنها توسط عملیات‌های بافتی قابل دسترسی است. عملیات‌های بافتی توسط واحدی خاص‌منظوره[۱۵] به نام واحد بافت[۱۶] انجام می‌شوند که در محاسبات گرافیکی بسیار کلیدی و پرکاربرد است. در محاسبات همه‌منظوره از حافظه بافت برای افزایش پهنای باند دسترسی به حافظه (با بهره بردن از حافظه نهان ویژه بافت) و یا درون‌یابی داده به‌صورت سخت‌افزاری (به کمک واحد بافت) استفاده می‌شود.

۲-۲-۳ ساختار هسته

هر هسته مجموعه دستورالعمل‌های شبه RISC معرفی شده در PTX 1.4 [۴۲] را پشتیبانی می‌کند. هر هسته دارای ساختار خطلوله[۱۷] SIMD است و همانطوری که در شکل ۲-۲ نشان داده شده است، ۷ مرحله متمایز دارد. در مرحله اول، وارپی از بین وارپ‌های آماده سیستم انتخاب می‌شود. در مرحله دوم، دستورالعمل

[14] Cache
[15] Fixed-function
[16] Texture unit
[17] Pipeline

بعدی این وارپ، که با شمارنده برنامه[۱۸] (PC) ی اختصاصی آن دنبال می‌شود، واکشی می‌شود و در مرحله سوم، این دستور دیکود می‌شود. در مرحله چهارم، عملوندهای ثباتی نخ‌های این وارپ از رجیسترفایل خوانده می‌شوند. در مرحله پنجم، نخ‌های فعال وارپ دستورالعمل دیکودشده را روی خطوط SIMD به‌صورت موازی اجرا می‌کنند. تعداد خطوط SIMD، پهنای SIMD گفته می‌شود و به‌طور متداول برابر ۸ [۳۴] یا ۱۶ [۴۷] خط است. هر خط SIMD گذردهی اجرای یک دستور ممیزشناور (توسط واحد FPU) یا یک دستور محاسبه و منطق (توسط واحد ALU) در هر سیکل را دارد. در صورتی که اندازه وارپ بزرگتر از پهنای SIMD باشد، اجرای وارپ چندین سیکل به‌طول می‌انجامد تا تمامی نخ‌های وارپ اجراشوند. به طور مثال اگر اندازه وارپ ۴ برابر پهنای SIMD است، اجرا ۴ سیکل به‌طول می‌انجامد. مرحله ششم ویژه دستورالعمل‌های دسترسی به حافظه است. دستورهای ذخیره/بارگذاری در/از حافظه سراسری/محلی/مشترک در این مرحله فعال هستند. اگر داده مورد نظر در حافظه نهان هسته یافت شد، در یک سیکل دسترسی انجام می‌شود. در غیر این صورت، درخواست برای ذخیره (یا بارگذاری) داده در (یا از) حافظه خارج تراشه به شبکه برتراشه فرستاده می‌شود. مرحله هفتم، برای دستورهای FPU، ALU، یا بارگذاری فعال است تا ثبات‌های خروجی دستور اجراشده توسط نخ‌های وارپ در رجیستر فایل پسنویسی شوند.

۲-۳ الحاق دسترسی‌ها به حافظه

هر هسته از قابلیت ویژه‌ای به نام الحاق بهره می‌برد که امکان ترکیب کردن چندین دسترسی به حافظه را فراهم می‌کند. الحاق در مرحله ششم خطلوله و حین اجرای دستورالعمل‌های ذخیره یا بارگذاری انجام می‌شود. در این مرحله، آدرس‌های دسترسی برای تعدادی نخ (که پهنای الحاق نامیده می‌شود) بررسی می‌شود و در صورتی که به یک بلوک حافظه نهان نگاشت شوند، ترکیب می‌شوند تا همزمان سرویس داده شوند. بدین ترتیب آدرس‌های درخواستی به تعدادی بلوک درخواستی کاهش پیدا می‌کنند که همه‌ی درخواست‌های یک بلوک همزمان سرویس داده می‌شود. اگر بلوک درخواستی در حافظه نهان وجود داشته باشد، درخواست در یک سیکل سرویس داده می‌شود و در غیر این صورت یک درخواست برای آن بلوک به شبکه برتراشه ارسال می‌شود. پهنای الحاق عموما برابر یک ربع وارپ، نصف وارپ، یا کل وارپ است.

[18] Program counter

الحاق حافظه قابلیتی کلیدی برای پردازنده‌های SIMT است که در آن ده‌ها هزار نخ همروند پهنای باند دسترسی به حافظه‌ی خارج از تراشه را به اشتراک می‌گذارند. پردازش وارپی در این پردازنده‌ها، فرصت الحاق را ایجاد و کارآمدی آن را بیشتر می‌کند زیرا همواره تعدادی زیادی نخ (به تعداد نخ‌های داخل وارپ) با هم همگام پیش می‌روند و همزمان به حافظه دسترسی پیدا می‌کنند.

۲-۴ مدیریت واگرایی

زمان‌بندی نخ‌ها در دانه‌بندی وارپ، پیچیدگی زمان‌بندی را کاهش می‌دهد زیرا تعداد المان‌های زمان‌بندی از $O(N)$ به $O(\frac{N}{M})$ کاهش پیدا می‌کند که N تعداد نخ‌های سخت‌افزاری پشتیبانی‌شده و M تعداد نخ‌ها در هر وارپ (که اندازه وارپ هم نامیده می‌شود) است. همچنین استفاده از وارپ، بهره‌برداری SIMD را بالا می‌برد زیرا همواره نخ‌های وارپ همگام نگه‌داشته می‌شوند و تضمین می‌شود که پهنای SIMD به طور موثر استفاده شود. آنچه استفاده از وارپ را با چالش مواجه می‌کند، پدیده‌های واگرایی انشعاب و واگرایی حافظه است که در ادامه این بخش به معرفی آنها پرداخته می‌شود.

۲-۴-۱ واگرایی حافظه

هنگام دسترسی به حافظه، بخشی از نخ‌های وارپ داده خواسته شده خود را در حافظه نهان پیدا می‌کنند و برخی دیگر با فقدان[19] مواجه می‌شوند. به این پدیده واگرایی حافظه می‌گویند. در این حالت کلیه نخ‌های وارپ منتظر سرویس‌دادن به همه فقدان‌ها می‌مانند. اگر هسته وارپ آماده‌ای برای اجرا نداشته باشد، گذردهی افت می‌کند در حالی که نخ‌هایی که با فقدان مواجه نشده‌اند می‌توانند ادامه پیدا کنند و تاخیر دیگر نخ‌ها را پنهان کنند.

۲-۴-۲ واگرایی انشعاب

نخ‌های یک وارپ، اگرچه یک دستور را اجرا می‌کنند ولی روی داده‌های خصوصی خود، که احتمالا با نخ‌های دیگر یکسان نیستند، پردازش انجام می‌دهند. در نتیجه هنگام اجرای دستورالعمل‌های انشعاب شرطی، ممکن است شرط برای نخ‌های یک وارپ یکسان سنجیده نشود. این اتفاق باعث می‌شود نخ‌های یک وارپ PCهای متفاوتی داشته باشند و ساختار یک «دستور برای تمام نخ‌ها» امکان‌پذیر نباشد. در این

[19] Miss

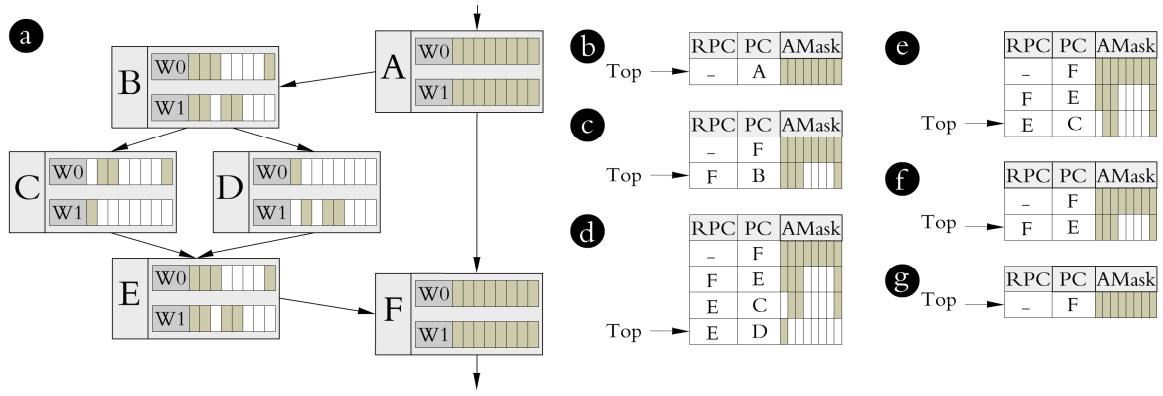
شمارنده برنامه	دنباله کد کرنل	مسیر نخ‌های وارپ
۱:	`a = threadIdx.x;`	↓↓↓↓↓↓↓↓
۲:	`if(a>4)`	↓↓↓↓↓↓↓↓
۳:	`{f += g*h;}`	---↓↓↓↓
۴:	`else{ f -= g*h;}`	↓↓↓↓↓---
۵:	`k[a] = f;`	↓↓↓↓↓↓↓↓

شکل ۲-۳: نمونه‌ای از رخ‌داد واگرایی انشعاب درون یک وارپ با ۸ نخ. نقطه ۵ پست‌دومیناتور بلافاصله برای انشعاب نقطه ۲ است.

حالت، نخ‌ها می‌توانند به PCهای متفاوتی واگرا شوند. به این پدیده واگرایی انشعاب می‌گویند. شکل ۲-۳ نمونه‌ای از واگرایی انشعاب درون یک وارپ را نشان می‌دهد.

در GPUهای امروزی، از روش همگرایی پشته‌ای[20] (SBR) برای مدیریت واگرایی انشعاب استفاده می‌شود. این روش PCهای مربوط به هر مسیر واگرایی[21] را به‌عنوان PCهای منطقی وارپ ذخیره می‌کنند و اجرای هر دسته از نخ‌های متناظر با این PCهای منطقی را متوالیا انجام می‌دهد. برای هر انشعاب شرطی یک نقطه بازهمگرایی[22] یا پُست‌دومیناتور[23] تعریف می‌شود تا مسیرهای واگرایی در آن نقطه دوباره بازهمگرا شوند. طبق تعریف نقطه P پست‌دومیناتور A است در صورتی که تمامی مسیرهای اجرایی از A تا انتهای برنامه از P عبور کنند [۳۷]. توجه شود که پست‌دومیناتور یکتا نیست. بدین جهت پست‌دومیناتور بلافاصله[24] تعریف می‌شود که یکتا است و طبق تعریف P پست‌دومیناتور بلافاصله A است اگر ۱) تمامی مسیرهای اجرایی از A تا انتهای برنامه از P عبور کنند و ۲) پست‌دومیناتور دیگری برای A در مسیری از A تا P وجود نداشته باشد. پست‌دومیناتور بلافاصله در زمان کامپایل از گرافِ کنترلِ جریانِ کرنل بدست می‌آید و آدرس آن به عنوانِ یکی از عملوندهای انشعابِ شرطی در کلمه دستور در باینری گنجانده می‌شود. روش SBR با تشکیل یک پشته، نخ‌های واگراشده را در پست‌دومیناتور بلافاصله دوباره بازهمگرا می‌کند تا دوباره «اجرای یک دستور برای تمام نخ‌های وارپ» ممکن شود.

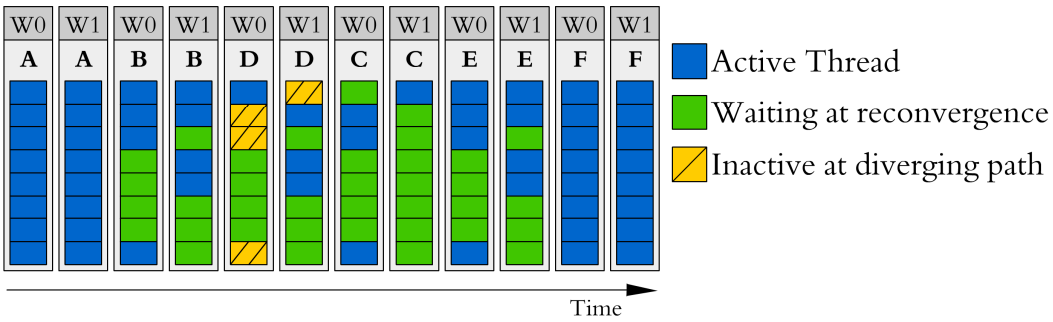
[20] Stack-based re-convergence
[21] Diverging path
[22] Re-convergence point
[23] Postdominator
[24] Immediate postdominator



شکل ۲-۴: مدیریت واگرایی انشعاب با روش SBR. فرض شده که ۸ نخ در هر وارپ وجود دارد و بالای پشته با Top نشان داده شده است.

روش SBR از یک پشته برای ذخیره کردن PCهای منطقی هر وارپ استفاده می‌کند و PC فیزیکی وارپ همان PCی منطقی بالای پشته است. هر سطر پشته، ردی از یک مسیر واگرایی نگه می‌دارد و سه میدان دارد: ۱) شمارنده برنامه (PC)، ۲) نخ‌های متناظر (ماسک فعال بودن)، و ۳) شمارنده برنامه نقطه بازهمگرایی (RPC). هنگام رخ دادن واگرایی انشعاب، PC موجود در سطر بالایی پشته با آدرس نقطه بازهمگرایی جایگزین می‌شود. بسته به اینکه انشعاب چند سو داشته (دو سو است وقتی بخشی از نخ‌ها به مسیر واگرایی X و بخشی دیگر به مسیر واگرایی Y پرش کنند و یک سو است وقتی بخشی از نخ‌ها به نقطه‌ی همگرایی و باقی به مسیر واگرایی X پرش کنند)، یک سطر (برای یک سویه) یا دو سطر (برای دو سویه) به پشته پوش می‌شوند تا مسیرهای واگرایی را در پشته ذخیره کنند. برای این سطرهای جدیدِ پوش‌شده، شمارنده برنامه‌ی ابتدای مسیرِ واگرایی در میدان PC و نخ‌های متناظر این مسیر در میدان «ماسک فعال بودن» ذخیره می‌شوند. همچنین آدرس بازهمگرایی در میدان RPC ذخیره می‌شود. ذخیره کردن RPC بدین جهت است که از اجرای نخ‌ها فراتر از آدرس بازهمگرایی جلوگیری شود تا نخ‌های مسیرهای واگرایی در نقطه بازهمگرایی همگام شوند. همان‌طوری که پیش‌تر گفته شد، میدان PCی بالاترین سطر پشته، شمارنده فیزیکی برنامه خواهد بود. پس از اینکه PC بالای پشته به RPC آن برسد، سطر بالایی پشته پاپ می‌شود و اجرا با دنبال کردن بالای پشته‌ی جدید ادامه پیدا می‌کند.

شکل ۲-۴، گراف کنترل جریان نوعی به همراه مسیرهای واگرایی نخ‌های درون دو وارپ (W0 و W1) را نشان می‌دهد. این شکل مثالی ارائه می‌کند تا واضح سازد که چگونه SBR واگرایی انشعاب را مدیریت



شکل ۲-۵: اجرای یک در میان وارپ‌های W0 و W1 در درون مسیرهای واگرایی. نخ‌های حاشورخورده، اگرچه می‌توانند فعال باشند، به علت نحوه‌ی مدیریت واگرایی انشعاب در SBR غیرفعال شده‌اند. اگر نخ‌های فعال در بلوک کد D دستور پرتاخیری اجرا کنند، وارپ آماده دیگری باقی نخواهد ماند. در زمان اجرا بلوک کد B، نیمی از نخ‌های روی نقطه بازهمگرایی منتظر هستند.

می‌کند. (a) گراف کنترل جریان برای یک کرنل است که واگرایی انشعاب درون ۲ وارپ را به نمایش می‌گذارد (W0 و W1). آخرین دستور از بلوک کد A، یک انشعاب واگرای یک سویه است. بلوک کد B با یک انشعاب دو سویه در دستور آخر به اتمام می‌رسد. واگرایی در بلوک کدهای A و B به‌ترتیب در F و E باز همگرا می‌شوند. (b) تا (g) نشان می‌دهند که SBR چگونه واگرایی انشعاب درون وارپ W0 را مدیریت می‌کند. همان‌طوری که در (b) نشان داده شده است، قبل از اجرای آخرین دستور A، تمامی نخ‌ها فعال هستند. هنگام اجرا کردن A، شمارنده برنامه (PC) بالای پشته به آدرس بازهمگرایی (RPC) تغییر پیدا می‌کند و سطر جدیدی پوش می‌شود که نشان‌دهنده‌ی مسیر واگرایی B است. نخ‌های دیگر روی F منتظر همگرا شدن هستند که در (c) نشان داده شده است. وارپ W0 نخ‌های موجود در B را اجرا می‌کند تا جایی که به انشعاب واگرا می‌رسد. سپس سطر بالای پشته به آدرس بازهمگرایی (E) تغییر پیدا می‌کند و دو سطر جدید پوش می‌شوند تا مسیرهای واگرایی و نخ‌های متناظرشان را نگه‌داری کنند، که در (d) دیده می‌شود. وارپ نخ‌های واگراشده در D را اجرا می‌کند. هنگامی که PC سطر بالای پشته به RPC برسد (یعنی با آن برابر شود)، سطر بالایی پاپ می‌شود و اجرا با نخ‌های سطر پایینی (بلوک کد C) ادامه پیدا می‌کند (f). همان‌طوری که در (g) نشان داده شده، نخ‌ها در بلوک کد F همگرا می‌شوند و اجرا با بیشترین بهره‌برداری از SIMD ادامه پیدا می‌کند.

در این مثال نشان داده شد که کاهش نخ‌های فعال در مسیرهای واگرایی باعث می‌شود برخی از خطوط SIMD بیکار بمانند. موقعیت ترکیب W0 و W1 در هنگام اجرای C وجود دارد تا خطوط بیکار کم شوند

و سیکل‌های اجرایی کل کم‌تر شود. نخ‌هایی که فعال نیستند یکی از دو وضعیت زیر را دارند: ۱) در مسیرهای واگرایی خود غیرفعال هستند و یا ۲) روی نقطه بازهمگرایی منتظر همگراشدن با باقی نخ‌های درون وارپ هستند. شکل ۲-۵ وضعیت کلیه نخ‌های مثال شکل ۲-۴ را در طول زمان نشان می‌دهد و مطالب فوق را ترسیم می‌کند. همانطوری که نشان داده شده، وضعیت نخ‌ها بین ۱) فعال، ۲) غیرفعال، و ۳) منتظر روی نقطه بازهمگرایی تغییر می‌کند. در سراسر این پژوهش از این طبقه‌بندی برای وضعیت نخ‌ها استفاده شده و به آن ارجاع می‌شود. در فصل بعدی نشان می‌دهیم که سیکل‌های بیکاری هسته مقداری قابل ملاحظه هستند و تعداد قابل توجهی نخ‌های غیرفعال و یا منتظر روی نقطه بازهمگرایی، می‌توانند کاندیدای مناسبی برای پوشاندن این سیکل‌های بیکاری باشند.

۲-۵ انگیزه

در این بخش مشاهدات کلیدی این پژوهش را ارائه می‌کنیم. در بخش اول انگیزه، سه اثر منفی واگرایی انشعاب بر گذردهی را معرفی و تحلیل می‌کنیم. در بخش دوم تاثیر اندازه وارپ بر ۱) الحاق، ۲) سیکل‌های بیکاری، و ۳) گذردهی را گزارش می‌کنیم. برای مشاهده مشخصات محک‌ها به جدول ۶-۱ مراجعه شود.

۲-۵-۱ تاثیر واگرایی انشعاب

روش SBR برای مدیریت واگرایی انشعاب، از ۳ جهت بهره‌برداری از SIMD را کاهش می‌دهد (که زمان اجرا از این جهات مستقیما افزایش پیدا می‌کند). این ۳ جهت عبارتند از: ۱) کاهش نرخ فعالیت SIMD، ۲) اجرای متوالی مسیرهای واگرایی، و ۳) منتظر نگه داشتن نخ‌هایی که به نقطه بازهمگرایی می‌رسند. در این بخش این ۳ مورد را عمیق‌تر معرفی می‌کنیم و رخ‌دادن آنها را از لحاظ کمی گزارش می‌کنیم.

۲-۵-۱-۱ نرخ فعالیت SIMD

بهره‌برداری SIMD تحت تاثیر دو عامل مهم است: ۱) میزان و الگوی رخ دادن واگرایی انشعاب، و ۲) وجود وارپ آماده برای اجرا. رخ دادن واگرایی انشعاب باعث بیکار ماندن برخی از خطوط SIMD می‌شود. از طرف دیگر، عدم وجود وارپ آماده برای اجرا، باعث بیکار ماندن کلیه خطوط SIMD می‌شود. عدم وجود وارپ آماده می‌تواند ناشی از تاخیر زیاد دسترسی به حافظه خارج هسته یا کمبود نخ در گرید باشد (که روش مدیریت واگرایی مسبب آن نیست). تمرکز این پژوهش بر واگرایی انشعاب است و لذا «نرخ

فعالیت SIMD» را به‌صورت زیر تعریف می‌کنیم تا معیاری باشد برای اندازه‌گیری بهره‌برداری از SIMD وقتی که تاثیر عدم وجود وارپ آماده از آن حذف می‌شود (سیکل‌هایی که تمام خطوط SIMD بیکار هستند نادیده گرفته می‌شوند):

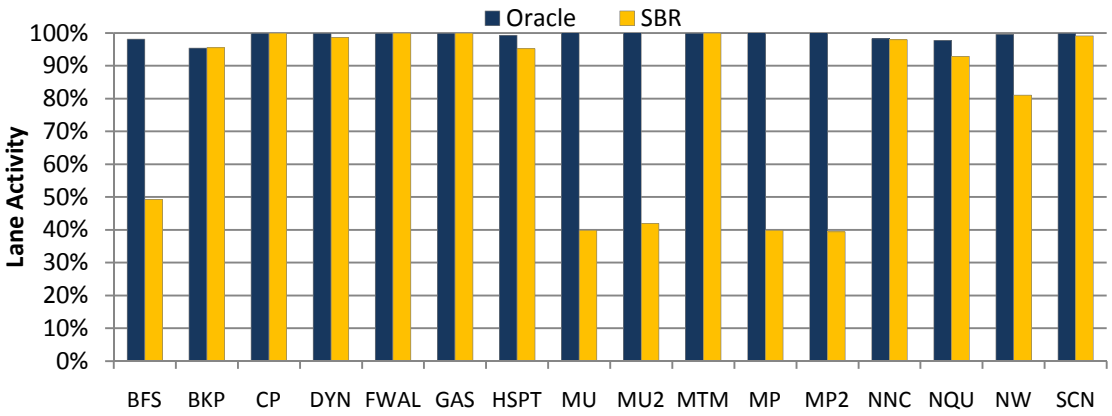
$$\text{Lane Activity} = 100 \times \frac{\sum_{i=1}^{n\_issues} act\_thds[i]}{n\_issues \times SIMDWidth} \qquad (۱-۲)$$

که در آن SIMDWidth برابر تعداد خطوط SIMD، n_issues تعداد سیکل‌هایی است که حداقل یک خط از SIMD فعال بوده است، و act_thds[i] تعداد خطوط فعال در سیکل iام است.

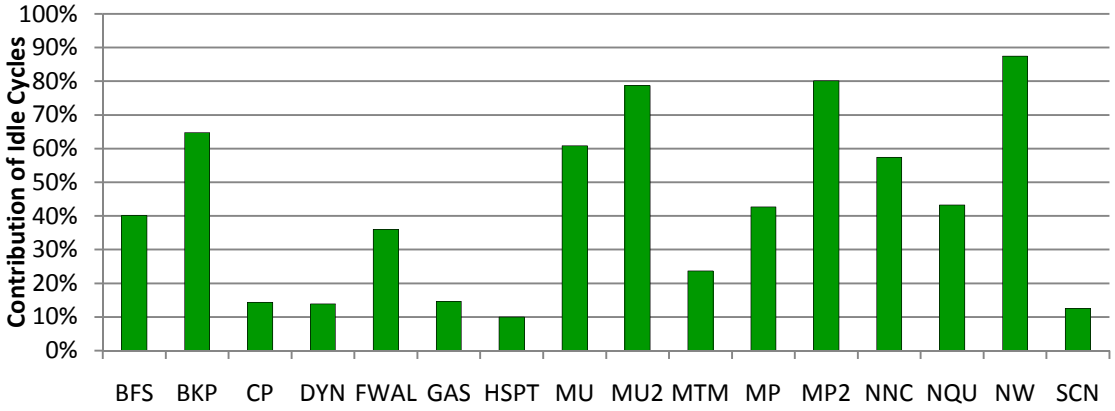
روش SBR مقداری از نرخ فعالیت SIMD می‌کاهد چون تنها نخ‌های یک وارپ می‌توانند همزمان ایشو شوند و همچنین نخ‌های واگرا شده‌ای را که به مسیر واگرایی بالای پشته تعلق ندارند، موقتا غیرفعال می‌کند. شکل ۲-۶، نرخ فعالیت SIMD برای محک‌های بررسی شده در این پژوهش گزارش می‌کند. برای هر محک ستون سمت راست، روش SBR و ستون سمت چپ روش پیشگو (Oracle) را گزارش می‌کند که در ادامه طریقه بدست آوردن آن شرح داده شده است. برای هر گرید، رد دستورالعمل‌هایی که هر نخ اجرا می‌کند، بدست می‌آید. سپس این رد به‌صورت استاتیک خوانده می‌شود و به‌صورت آفلاین دستورالعمل‌های نخ‌ها در وارپ‌هایی گروه‌بندی می‌شود. دستورالعمل‌های گروه‌بندی شده از رد حذف می‌شوند و این کار ادامه پیدا می‌کند تا تمامی دستورالعمل‌های موجود در رد با وارپ پوشش داده شوند. نرخ فعالیت SIMD ای که از وارپ‌بندی‌های این روش پیشگوی غیرواقعی بدست می‌آید، کران بالایی از نرخ فعالیت SIMD است که توسط مکانیزم‌های گروه‌بندیِ پویای وارپ می‌تواند بدست آید. همانطوری که مشاهده می‌شود، نرخ فعالیت SIMD، در برخی از محک‌ها بسیار عدد کوچکی است.

۲-۵-۱-۲ اجرا متوالی مسیرهای واگرایی

زمانی که هسته وارپ آماده‌ای برای اجرا نداشته باشد، گذردهی افت می‌کند. این اتفاق زمانی رخ می‌دهد که نخ‌های فعال در بالای پشته دستور العملی با تاخیر زیاد اجرا کنند و وارپ دیگری برای پوشاندن این تاخیر وجود نداشته باشد. شکل ۲-۷، درصد سیکل‌های بیکاری هسته در کل سیکل‌هایی که هسته وارپ ناتمامی داشته را نشان می‌دهد. همانطوری که گزارش شده، هسته تا ۹۳٪ (در NW) از سیکل‌ها می‌تواند بیکار باشد.

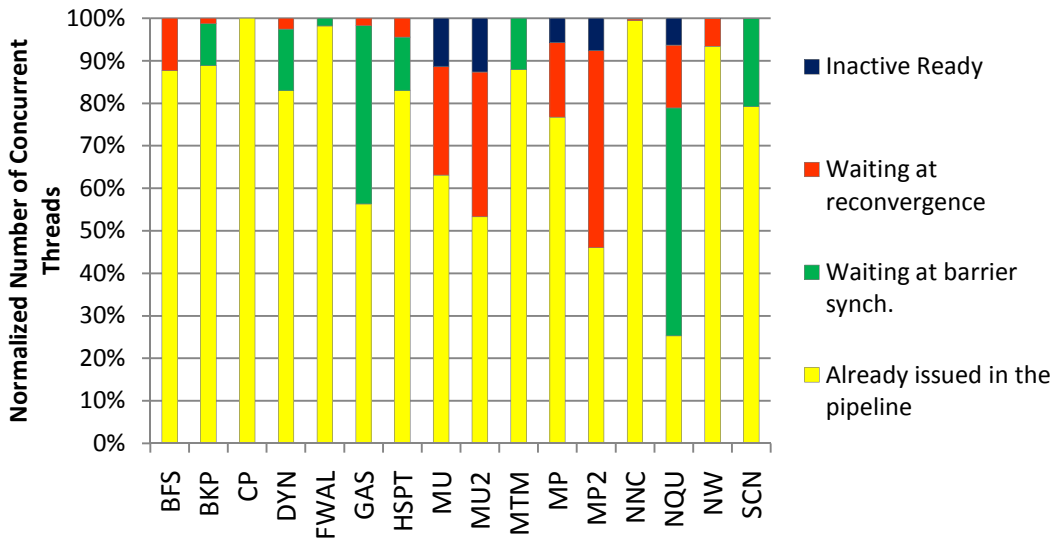


شکل ۲-۶: تاثیر واگرایی انشعاب بر نرخ فعالیت SIMD برای روش‌های SBR و پیشگو (Oracle).



شکل ۲-۷: درصد رخ دادن وضعیتی که هسته به‌علت عدم وجود وارپ آماده متوقف شود. اعداد برای اندازه وارپ ۳۲ نخ گزارش شده است.

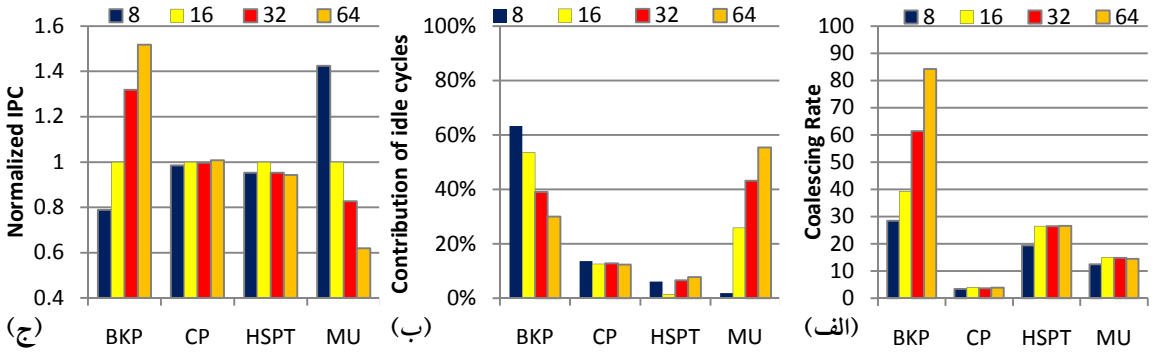
تعداد نخ‌های غیرفعال با افزایش اندازه وارپ و تعداد انشعاب‌های شرطی تودرتو افزایش پیدا می‌کند. شکل ۲-۸، تفکیک وضعیت نخ‌ها در سیکل‌هایی که هسته بیکار بوده است را نشان می‌دهد. هنگامی که هسته بیکار است، نخ‌های همروند آن یکی از ۴ وضعیت زیر را دارند: ۱) در حال اجرای دستوری در درون خطلوله هستند (شامل دستورالعمل‌های حافظه)، ۲) روی نقطه همگرایی منتظر همگام‌شدن با نخ‌های واگرا شده‌ی دیگر هستند، ۳) روی همگام کننده‌های درون بلوکی منتظر رسیدن باقی نخ‌ها هستند، و یا ۴) در سطرهای پایین پشته هستند و غیرفعال شده‌اند. همان‌طوری که در شکل ۲-۸ نشان‌داده شده است، مقادیری از نخ‌ها در سطرهای پایین پشته غیرفعال شده‌اند. با افزودن نخ‌های غیرفعال به مجموعه وارپ‌های آماده می‌توان درصدی از سیکل‌های بیکاری هسته را پوشاند.



شکل ۲-۸: تفکیک وضعیت نخ‌های همروند در زمانی‌هایی که هسته بیکار بوده است.

۲-۵-۱-۳ انتظار روی نقطه همگرایی

روش SBR نیازمند است که دسته از نخ‌های واگراشده را که زودتر به نقطه بازهمگرایی می‌رسند، منتظر نگه‌دارد تا با دیگر مسیرهای واگرایی همگام شوند. وقتی که مسیرهای واگرایی یک انشعاب متوازن نیستند، این دوره انتظار می‌تواند بسیار طولانی باشد. متوازن نبودن مسیرهای واگرایی به معنی آن است که نخ‌های مسیرها ممکن است تعداد دستورهای متفاوتی اجرا کنند و تاخیر متفاوتی را تجربه کنند. شکل ۲-۸ نشان می‌دهد که تا ۲۶٪ (برای محک BFS) از نخ‌های بیکار می‌توانند روی نقطه همگرایی منتظر باشند در حالی که هسته بدون وارپ آماده مانده است. این میزان نخ منتظر، می‌توانند درصد قابل توجهی از سیکل‌های بیکاری را پنهان کنند. مشاهده کلیدی آن است که به تمام نخ اجازه عبور از نقطه همگرایی دادن، باعث کاهش نرخ فعالیت SIMD می‌شود. علت آن است که نخ‌هایی که از نقطه همگرایی عبور می‌کنند، ممکن است دیگر همگرا نشوند و بهره‌برداری کم از SIMD در سراسر کرنل فراگیر شود. راه دیگر آن است که تعداد کمتری نخ روی نقطه همگرایی همگام شوند. تنها منتظر نگه‌داشتن نخ‌هایی الزامی است که عبور دادن آنها نرخ فعالیت SIMD را کاهش می‌دهد (مثلا نگه داشتن تعداد نخ‌هایی برابر با پهنای SIMD می‌تواند کافی باشد).



شکل ۲-۹: الف) نرخ الحاق، ب) درصد سیکل‌های بیکاری، و ج) گذردهی برای ماشین‌هایی با اندازه وارپ متفاوت. گذردهی به ماشینی با اندازه ۱۶ نخ در وارپ نرمال شده است.

۲-۵-۲ تاثیر اندازه وارپ

زمانی که الحاق دسترسی‌ها به حافظه روی کل وارپ (بین تمامی نخ‌های وارپ) انجام می‌شود، اندازه وارپ تاثیر دوگانه‌ای روی گذردهی دارد. در این بخش با فرض معماری پایه‌ای که الحاق دسترسی‌ها به حافظه را روی کل وارپ انجام می‌دهد، تاثیر اندازه وارپ بر تعداد سیکل‌های بیکاری هسته، الحاق دسترسی به حافظه، و گذردهی را گزارش می‌کنیم. نرخ فعالیت SIMD گزارش نشده است زیرا مشاهدات ما نشان می‌دهد که اندازه وارپ تاثیر بسیار ناچیزی (کمتر از ۱٪) روی نرخ فعالیت SIMD دارد. در این بخش، نتایج را برای نماینده‌هایی از محک‌ها (از گونه رفتارهای متفاوت) گزارش می‌کنیم و نتایج کامل را در بخش ۶-۴ ارائه می‌کنیم.

۲-۵-۲-۱ الحاق دسترسی‌ها به حافظه

دسترسی‌های حافظه برای نخ‌های درون یک وارپ به تعداد کمتری تراکنش حافظه الحاق می‌شود تا تقاضای پهنای باند حافظه را کم کند. برای اندازه‌گیری الحاق دسترسی به حافظه، نرخ الحاق را به‌صورت زیر تعریف می‌کنیم:

$$\text{Coalescing rate} = \frac{Total\ memory\ insntruction}{Total\ of\ fchip\ request} \qquad (۲-۲)$$

در باقی پژوهش از این تعریف برای تخمین کارآمدی پردازنده در الحاق دسترسی‌ها به حافظه استفاده می‌کنیم. شکل ۲-۹(الف) نرخ الحاق را برای ماشین‌هایی با اندازه وارپ‌های متفاوت گزارش می‌کند. همان‌گونه که نمایش‌داده شده است، افزایش اندازه وارپ، نرخ الحاق را بهبود می‌بخشد. افزایش اندازه

وارپ احتمال اینکه آدرس‌های دسترسی به حافظه از سوی نخ‌های یک وارپ در یک بلوک نهان قرار بگیرد را افزایش می‌دهد. این افزایش برای اندازه وارپ‌های بزرگتر از ۳۲ کاهش می‌یابد زیرا عملا یک بلوک نهان (۱۶ کلمه‌ی ۳۲ بیتی) کوچکتر از آن است که عملوند تمامی این نخ‌ها را شامل شود و اشباع می‌شود. متعاقبا بزرگتر کردن اندازه وارپ بیش از این اندازه، سود کمی برای الحاق دارد.

۲-۵-۲-۲ سیکل‌های بیکاری

شکل ۲-۹(ب) نرخ رخ‌دادن سیکل‌های بیکاری برای GPUهایی با اندازه وارپ‌های متفاوت را گزارش می‌کند. سیکل‌های بیکاری ناشی از وارپ‌های همروند، واگرایی حافظه، و واگرایی انشعاب هستند. وارپ‌های کوچک سیکل‌های بیکاری را کم می‌کنند، زیرا واگرایی انشعاب و واگرایی حافظه در آنها کمتر رخ می‌دهد (مانند محک MU). از طرف دیگر، برای برخی دیگر از محک‌ها (مانند BKP)، وارپ‌های کوچک بسیاری از دسترسی‌های قابل الحاق را از دست می‌دهند که فشار حافظه را افزایش می‌دهد. این فشار، تاخیر دسترسی به حافظه را بالا می‌برد که خود باعث افزایش سیکل‌های بیکاری می‌شود (مانند BKP).

۲-۵-۲-۳ گذردهی

افزایش اندازه وارپ می‌تواند اثر عکس روی گذردهی داشته باشد. گذردهی می‌تواند بهبود یابد اگر سود بدست آمده از الحاق دسترسی‌ها به حافظه، سربار وارپ بزرگ را جبران کند. سربار وارپ بزرگ، همگام کردن‌های متوالی روی تمام دستورالعمل‌ها است (خصوصاً دستورالعمل‌های حافظه هنگام واگرایی حافظه) که خود می‌تواند منجر به سیکل‌های بیکاری شود. از طرف دیگر، گذردهی می‌تواند کاهش یابد اگر سربار همگام کردنی که وارپ‌های بزرگ ایجاد می‌کنند، بر سود الحاق دسترسی‌ها به حافظه ناشی از آن سنگینی کند. شکل ۲-۹(ج) گذردهی برای GPUهایی با اندازه وارپ‌های متفاوت را گزارش می‌کند. همانطوری که گزارش شده است، اندازه وارپ اثر متفاوتی روی گذردهی دارد؛ گذردهی در BKP با اندازه وارپ بیشتر می‌شود، در MU با اندازه وارپ کاهش می‌یابد، و در HSPT با اندازه وارپ میانه (۱۶ نخ) بهترین گذردهی بدست می‌آید. این درحالی است که وابستگی CP به اندازه وارپ بسیار ناچیز است. از مشاهدات بدست آمده نتیجه می‌گیریم که اندازه وارپ می‌تواند گذردهی را به شکل‌های متفاوتی تحت تاثیر قرار دهد.

فصل ۳

کارهای پیشین

لی و همکاران [۳۰] کارایی یک پردازنده چندهسته‌ای در مقایسه با یک GPU همرده را برای طیف گسترده‌ای از کاربردها ارزیابی می‌کنند و نشان می‌دهند در صورتی که کد در هر دو پلتفورم بهینه شود به طور متوسط GPU حدود ۲.۵ برابر سریع‌تر از یک پردازنده چندهسته‌ای کار می‌کند. در سال‌های اخیر تحقیقات گسترده‌ای برای ارزیابی، مدل‌سازی، بهبود انرژی مصرفی، و بهبود کارایی در معماری‌های شتاب‌دهنده‌های بسیار هسته‌ای[1] و شبیه به GPU صورت گرفته است که در مروری بر برخی از آنها می‌کنیم.

۳-۱ ارزیابی

برای ارزیابی کارایی GPUها در محاسبات همه‌منظوره، مجموعه محک‌های استانداردی وجود دارد که عبارتند از NVIDIA CUDA SDK [۴۱]، RODINIA [۵]، SHOC [۹]، و Parboil [۴۵]. کر و همکاران [۲۳] بیش از ۵۰ کرنل کودا را ارزیابی کرده‌اند و رفتارهایی از قبیل اهمیت همگرایی و اشتراک داده بین نخ‌ها را مشاهده کرده‌اند. افزون بر این، با معرفی برخی از معیارهای جدید (از جمله نرخ فعالیت[2])، کرنل‌ها را رفتارسنجی کرده اند. لشگر و بنی‌اسدی [۲۷] با معرفی مدل‌های ماشینی کارایی GPU را در مقایسه با GPUی ایده‌آل سنجیده‌اند. ماشین‌های ایده‌آل با ۱) صفر کردن تاخیر دسترسی به حافظه، ۲) از بین بردن واگرایی انشعاب (تبدیل SIMD به MIMD)، و ۳) بی‌نهایت کردن درجه چندوارپی نشان می‌دهند که ایده‌آل شدن هریک از این سه پارامتر چه میزان بهبود کارایی GPUهای امروزی کمک می‌کند و البته چه میزان از ماشین کاملا ایده‌آل به دور است. بِلِم و همکاران [۳] طیف بزرگی از کاربردها را روی GPU ارزیابی می‌کنند و آنها را براساس گلوگاه کارایی‌شان طبقه‌بندی می‌کنند. آنها شش گلوگاه‌ها کارایی تعریف

[1] Many-core accelerators
[2] Activity factor

می‌کنند که عبارتند از: ۱) تعداد بلوک‌های درون گرید، ۲) تعداد نخ‌های درون بلوک، ۳) تعداد نخ‌های درون وارپ، ۴) اجرای سریال وارپ به علت واگرایی انشعاب، ۵) پهنای باند حافظه، و ۶) تاخیر دسترسی به حافظه.

۳-۲ مدل‌سازی

هونگ و کیم [۱۹] مدلی ریاضی برای مدل کردن گذردهی GPU ارائه کرده‌اند. ورودی مدل مذکور ویژگی بارکاری (از قبیل تعداد) و پارامترهای معماری است. این مدل برای بارکاری‌هایی که واگرایی انشعاب کمی دارند دقیق‌تر عمل می‌کند. ژیا و همکاران [۲۱] نشان می‌دهند که فضای طراحی GPUها بسیار بزرگ است و برای پیدا کردن نقطه پیکربندی بهینه در این فضا بیش از ۱۰ هزار پیکربندی تاثیرگذار باید ارزیابی شوند. سپس مدلی مبتنی بر تحلیل رگرسیون ارائه می‌کنند که با داشتن زمان اجرای کاربرد در ۳۰۰ نقطه تصادفی از این فضا، با دقت قابل قبولی (خطای کمتر از ۴٪)، زمان اجرا کاربرد در کلیه نقاط این فضا را پیش‌بینی می‌کنند. اگرچه آنها مدل را برای پیش‌بینی زمان اجرا ارزیابی می‌کنند، ولی این مدل رگرسیونی می‌تواند برای تخمین توان مصرفی و یا هر پارامتر دیگری بکار گرفته شود.

برای مدل‌سازی عملکرد GPU، ابزارهایی برای شبیه‌سازی دستورالعمل‌های PTX [۴۲]، کودا [۳۹]، و OpenCL [۲۴] ارائه شده است. از شاخص‌ترین این ابزارهای می‌توان به GPGPU-sim [۲]، Ocelot [۱۱] و Barra [۸] اشاره کرد. ابزار GPGPU-sim شبیه‌سازی برای مدل‌سازی سیکل به سیکل اجرای کرنل روی پردازنده‌ای شبیه به GPU است. این ابزار اجرای بارکاری‌های کودا و OpenCL را مدل می‌کند و گزارشی از اجرای بارکاری روی GPU برمی‌گرداند که حاوی تعداد سیکل‌های اجرایی، تعداد دستورالعمل‌های اجرا شده، گذردهی، و برخی دیگر از شمارنده‌های جزئی که از اتفاقات ماجول‌های معماری از جمله حافظه نهان، کنترل کننده حافظه، و DRAM گزارش می‌دهد. ابزار Ocelot شامل مجموعه‌ای از ابزارها برای تحلیل و ارزیابیِ بهینه‌سازی‌های کامپایلری و شبیه‌سازی بارکاری‌های کودا روی GPUهای NVIDIA، GPUهای ATI، CPUهای SIMD، و CPUهای چندهسته‌ای است. در نسخه‌های اخیر امکان شبیه‌سازی معماری GPU در این ابزار کمرنگ‌تر شده است و این ابزار وقف بهینه‌سازی‌های کامپایلری و ترجمه باینری‌های PTX به زبان میانه LLVM [۲۹] شده که این زبان میانه امکان تبدیل شده به باینری‌های، PTX، CAL [۱]، و x86 را دارد.

۳-۳ بهبود انرژی مصرفی

استفاده از تمامی هسته‌های محاسباتی الزاما بهترین کارایی را بدست نمی‌دهد. علت آن است که در بارکاری‌های وابسته به پهنای باند حافظه، ازدیاد تعداد هسته‌های محاسباتی نه تنها باعث بهبود کارایی نمی‌شود بلکه باعث افزایش تراکم روی حافظه خارج تراشه می‌شود که خود تاخیر دسترسی به حافظه را بالا می‌برد. بدین ترتیب در برخی از کاربردها با افزایش تعداد هسته‌های فعال، نسبت گذردهی به توان مصرفی کاهش پیدا می‌کند. هونگ و کیم [۲۰] با مشاهده کردن این پدیده، سیستمی برای کنترل تعداد هسته‌ها فعال ارائه می‌کنند تا نسبت $\frac{\text{زمان اجرا}}{\text{توان مصرفی}}$ بیشینه نگه داشته شود. این تکنیک، بهینه تعداد هسته‌ها را برای رسیدن به بیشینه $\frac{\text{زمان اجرا}}{\text{توان مصرفی}}$ پیشبینی می‌کند که این کار با تخمین زمان اجرا و توان مصرفی انجام می‌دهد.

لی و همکاران [۳۲] اثر تغییرات فرآیند ساخت[3] بر معماری GPU را ارزیابی می‌کنند. مشاهدات نشان می‌دهد که اندازه هر هسته مساحت کوچکی از تراشه را می‌گیرد و به دنبال آن فرکانس کاری هسته‌ها می‌توانند بسیار متفاوت باشد. آنها با درنظر گرفتن معماری پایه‌ای که تمامی هسته‌ها با یک کلاک (کندترین) کار می‌کنند، دو تکنیک متفاوت برای بهبود کارایی با تغییر کلاک‌دهی را ارزیابی می‌کنند. تکنیک اول به هر هسته کلاکی مستقل می‌دهد به نحوی که هر هسته با بیشترین کلاک خود کار کند. تکنیک دوم، n هسته از کندترین هسته‌ها را خاموش می‌کند و سپس همگی هسته‌های فعال باقیمانده را مشترکا با کندترین کلاک می‌راند (نتایج برای n از ۱ تا تمام هسته‌ها جاروب شده است).

گِبهارت و همکاران [۱۶] دو طراحی متعامد برای بهبود انرژی مصرفی در GPU ارائه می‌کنند. در طراحی اول با نهان کردن برخی از ثبات‌ها که متداولا استفاده می‌شوند، از دسترسی به رجیسترفایل بزرگ جلوگیری می‌شود. این طراحی بر این مشاهده استوار است که رجیسترها بعد از نوشته شدن، اغلب در کمتر از ۳ دستور بعدی برای آخرین بار خوانده می‌شوند. با بهره گرفتن از این رفتار، رجیسترها به‌طور موقت در حافظه کوچکتری ذخیره می‌شوند زیرا به زودی برای آخرین بار استفاده خواهند شد. بدین

[3] Process variation

ترتیب، تعدادی از دسترسی‌ها به رجیسترفایل بزرگ کم می‌شود و از آنجایی که رجیسترها در حافظه‌ای بسیار کوچکی نهان می‌شوند، میزانی از توان مصرفی فرآیند دسترسی به عملوندهای رجیسترفایل کم می‌شود. ارزیابی‌ها نشان می‌دهد که طراحی اول می‌تواند تا ۳۶٪ انرژی مصرفی ساختار رجیسترفایل را کم کند. در طراحی دوم، زمان‌بندِ وارپِ دوسطحی را معرفی می‌کنند که هدف از آن کاهش سربار (انرژی) زمان‌بندی است. زمان‌بند سطح نخست، تعدادی کمی وارپ را نگه‌داری می‌کند و مادامی که وابستگی عملوند تمامی این وارپ‌ها را متوقف نکرده است، انتخاب و زمان‌بندی از میان این وارپ‌ها انجام می‌شود. در صورت توقف وارپ‌های سطح اول، از وارپ‌های سطح دوم بهره گرفته می‌شود و تعدادی از وارپ‌های سطح دوم جای وارپ‌های سطح اول را می‌گیرند. کولانژ و همکاران [۷] نشان داده‌اند که میزان چشمگیری از مقادیر عملوندهای نخ‌های همروند در GPUها یکسان هستند. با مشاهده این رفتار، آنها طراحی ترکیبی برای رجیسترفایل پیشنهاد می‌کنند که رجیسترفایل عددی[۴] را در کنار رجیسترفایل برداری[۵] قرار می‌دهد. ثبات‌هایی که مقدار آنها برای دسته‌ای از نخ‌ها یکسان است از رجیسترفایل عددی خوانده می‌شوند و در صورت تمایز، ثبات‌ها در رجیسترفایل برداری نگه‌داری می‌شوند. ارزیابی‌های آنها نشان می‌دهد که این طراحی به‌طور متوسط ۱۹٪ از دسترسی‌های خواندن و ۱۱٪ از دسترسی‌های نوشتن به رجیسترفایل برداری کم می‌کند.

داسیکا و همکاران [۱۰] نشان داده‌اند که پهنای SIMD روی نسبت گذردهی به توان مصرفی بسیار موثر است و پهنای ۳۲ خط را برای بارکاری‌های علمی ارزیابی شده بهینه یافتند. مِنگ و همکاران [۳۶] نیز مشاهده‌ای مشابه داشته‌اند و عمق چندنخی را هم بر این نسبت موثر دیده‌اند. عمق چندنخی برابر با تعداد نخ‌های همروندی است که روی هر هسته اجرا می‌شوند. آنها تکنیکی معرفی می‌کنند تا با تغییر عمق چندنخی و پهنای SIMD، مقدار بهینه‌ی آنها برای رسیدن به بیشترین گذردهی در هر کاربرد به‌صورت پویا پیدا شود. با فرض وجود پشتیبانی سخت‌افزاری برای باز پیکربندی این دو پارمتر، برای انتخاب مقدار بهینه عمق چندنخی و پهنای SIMD، دو راه حل پیشنهاد می‌کنند: نرم‌افزاری و سخت‌افزاری. در روش نرم‌افزاری تعیین پارامترهای مذکور از طریق فراخوانی‌های تابع در اختیار برنامه‌نویس قرار داده می‌شود و او با شناخت خوبی که از برنامه دارد این پارامترها را به بهترین عدد انتصاب می‌کند. در روش سخت‌افزاری، سخت‌افزار

[4] Scalar
[5] Vector

در یک فاز کوچک (که نسبت به کل زمان اجرا ناچیز است) با اجرای بخشی از کاربرد، مقدار بهینه این پارامترها را می‌یابد و باقی اجرا را با پارامترهای جدید ادامه می‌دهد.

۳-۴ کارایی

لاکشمینارایانا و کیم [۲۶] سیاست‌های متفاوتی از زمان‌بندی وارپ و زمان‌بندی درخواست‌های حافظه را ارزیابی کرده‌اند. آنکه کاربردها را به دودسته متقارن و نامتقارن تقسیم می‌کنند و نشان می‌دهند که سیاست‌های عدالتی برای کاربردهای متقارن کارایی را بهبود می‌دهد در حالی که برای کاربردهای نامتقارن به سیاست‌های پیچیده‌تری نیاز است. یوآن و همکاران [۵۰] سیاست زمان‌بندی درخواست‌ها به DRAM را ارزیابی می‌کنند و نشان می‌دهند که سیاست خارج از ترتیب[۶] بهترین کارایی را دارد. از طرفی، برای پردازنده‌های بسیار هسته‌ای که تعداد درخواست‌ها به حافظه زیاد است، پیاده‌سازی این زمان‌بندی به‌دلیل مصرف انرژی زیاد پرهزینه است. مشاهده‌ی دیگر آنها نشان می‌دهد که درخواست‌های حافظه به میزان قابل‌توجهی محلیت ردیف در DRAM دارند و این محلیت توسط مسیریاب‌های داخل شبکه میان ارتباطی برهم می‌خورد. سپس با ارائه طراحیِ کم هزینه‌ای برای مسیریاب‌های شبکه میان ارتباطی، سعی می‌کنند که محلیت درخواست‌ها را پیش از رسیدن به صف DRAM بهبود دهند و با زمان‌بند FIFO به کارایی‌ای نزدیک به زمان‌بند خارج از ترتیب برسند.

ژانگ و همکاران [۵۱] چارچوب[۷] نرم‌افزاری معرفی می‌کنند که با استفاده از سه تکنیک جابه‌جایی داده[۸]، تعویض شغل[۹]، و تبدیل پیوندی[۱۰] سعی در حداقل کردن بی‌نظمی‌های دسترسی به حافظه و واگرایی انشعاب می‌کنند. تکنیک جابه‌جایی داده چینش جدیدی برای المان‌های آرایه‌ی چیده شده در حافظه ایجاد می‌کند و برای کاهش بی‌نظمی‌های دسترسی به حافظه پیشنهاد شده است. تکنیک تعویض شغل وظایف نخ‌ها را جابه‌جا می‌کند تا بی‌نظمی دسترسی به حافظه و واگرایی انشعاب کمتر شود. وظایف نخ‌ها عموما از شناسه‌ی نخ مشخص می‌شود و تکنیک تعویض شغل شناسه‌ی مجازی به نخ‌ها می‌دهد تا وظایف آنها تعویض شود.

[6] Out-of-order
[7] Framework
[8] Data reordering
[9] Job swapping
[10] Hybrid transformation

کولانژ در [۶] به مقایسه دقیقی بین روش‌های متفاوت مدیریت انشعاب در پردازنده‌های SIMD می‌پردازند و شاخص‌ترین آنها را معرفی می‌کند. برای رعایت اختصار از تکرار روش‌های قدیمی‌تر خودداری می‌کنیم و خواننده را به [۶] ارجاع می‌دهیم. در ادامه این بخش مروری می‌کنیم بر کارهای انجام شده برای بهبود مدیریت انشعاب در GPUها که درسال‌های اخیر پیشنهاد شده‌اند. تحقیقات گذشته دو نوع راه‌حل برای بهبود نرخ فعالیت SIMD پیشنهاد کرده‌اند: ۱) همگرا کردن نخ‌های یک وارپ در نقطه‌ای زودتر از پست‌دومیناتور بلافاصله [۱۵، ۱۲]، یا ۲) استفاده از نخ‌های دیگر وارپ‌ها برای پرکردن خطوط غیرفعال و تشکیل یک وارپ جدید [۱۵، ۱۳، ۳۸]. دو نوع راه حل برای کاهش زمان انتظار روی نقطه همگرایی پیشنهاد شده است: ۱) از بین بردن تمام همگام شدن‌ها روی نقطه بازهمگرایی [۱۳]، ۲) عبور دادن انتخابی نخ‌ها منتظر روی نقطه همگرایی [۳۵].

فانگ و همکاران [۱۳] روش DWF [11] را پیشنهاد داده‌اند تا بهره‌برداری کم منابع محاسباتی هنگام رخ دادن انشعاب را بهبود دهند. مکانیزم DWF نرخ فعالیت SIMD را با استفاده از گروه‌بندی پویا هدف قرار می‌دهد تا نخ‌های واگرا شده از وارپ‌های مختلف را در یک وارپ قرار دهد. هنگام رخ دادن واگرایی انشعاب، وارپ واگرا شده منحل می‌شود و دو وارپ جدید ساخته می‌شوند که هریک شامل نخ‌های یک مسیرواگرایی‌اند. وارپ‌های جدید در یک جدول نگه داشته می‌شوند و تا در صورت امکان با نخ‌های دیگر وارپ‌ها پس از واگرایی ترکیب شوند و وارپی با نخ‌های فعال بیشتری بسازند. DWF مکانیزمی برای اعمال حتمی همگرایی ندارد و با استفاده از سیاست زمان‌بندی وارپ‌ها سعی در همگرا کردن وارپ‌ها دارد. آنها سیاست‌های زمان‌بندیِ وارپ گوناگونی را ارزیابی کرده‌اند که از میان آنها می‌توان به کوچکترین شمارنده برنامه[12]، شمارنده‌ی برنامه غالب[13]، و شمارنده‌ی برنامه اقل[14] اشاره کرد.

مِنگ و همکاران [۳۵] DWS [15] را پیشنهاد کرده‌اند تا موازی‌سازی در سطح حافظه (MLP [16]) را بهبود دهند و پوشاندن تاخیر را با تعداد کمتری نخ همروند انجام دهند. روش DWS واگرایی انشعاب و واگرایی حافظه را هدف قرار می‌دهد و با برش وارپ هنگام رخ‌دادن واگرایی از غیرفعال شدن نخ‌ها جلوگیری می‌کند. برش وارپ در هنگام واگرایی انشعاب باید به‌گونه‌ای باشد که وارپ‌ها دوباره همگرا شوند. ساختار

[11] Dynamic Warp Formation
[12] Minimum PC
[13] Majority PC
[14] Minority PC
[15] Dynamic warp subdivision
[16] Memory Level Parallelism

معماری DWS به گونه‌ای است که تنها همگام شدن روی نقطه‌ی بازهمگرایی متناظر با اولین برش تضمین می‌شود و در نتیجه برش‌های متوالی یک وارپ ممکن است باعث شود بخش اعظمی از کرنل با وارپ‌های بریده‌شده‌ی باریک اجرا شود که نرخ فعالیت SIMD را کم می‌کند. از طرف دیگر، برش وارپ در هنگام واگرایی حافظه تنها زمانی سودمند است که جلو رفتن نخ‌هایی که داده خود را در حافظه نهان یافته‌اند داده‌ای برای دیگر نخ‌های منتظر پیش‌واکشی[17] کند. در غیر این صورت، تنها نرخ فعالیت SIMD کمتر می‌شود. روش DWS سه مکانیزم مکاشفه‌ای (که به آنها روش‌های زیربخش کردن[18] گفته شده) ارائه می‌کند تا موقعیت مناسب برای برش وارپ را تخمین بزند.

روش TBC [19] [۱۵] از پشته همگرایی استفاده می‌کند با این تفاوت که یک پشته عریض‌تر و برای هر بلوک نخ مشترکا استفاده می‌شود به جای آنکه هر وارپ، یک پشته داشته باشد. به عبارت دیگر در TBC پهنای پشته به پهنای تعداد نخ‌های درون بلوک است. ساختارِ کارکردِ پشته کاملا مشابه روش SBR است. اما TBC ماسک فعالیت سطر بالای پشته را فشرده کرده و چندین زیروارپ مستقل (برابر پهنای SIMD) ایجاد می‌کند. این مکانیزم باعث می‌شود در مسیرهای واگرایی که بالای پشته نخ‌های فعال کمتری دارد، خطوط خالی زیروارپ‌ها با آوردن نخ‌هایی از زیروارپ‌های دیگر پر شود و نرخ فعالیت SIMD بالا برود. این زیروارپ‌ها در هنگام واگرایی وارپ بزرگ ساخته می‌شوند و در جدول زیروارپ‌ها قرار می‌گیرند و مستقل از یکدیگر زمان‌بندی می‌شود. این زیروارپ‌ها هنگامی که به انشعاب شرطی یا نقطه همگرایی برسند، متوقف می‌شوند تا با دیگر زیروارپ‌هایی یک بلوک نخ همگام شوند.

ناراسیمن و همکاران [۳۸] روش LWM [20] را پیشنهاد می‌کنند تا نرخ فعالیت SIMD را بهبود دهند. LWM به‌صورت استاتیک نخ‌ها را در وارپ‌های بزرگی گروه‌بندی می‌کند و هر وارپ‌ها را در چندین سیکل اجرا می‌کند. در مسیرهای واگرایی که تعداد نخ‌های فعال بالای پشته کم می‌شوند، برخی از سیکل‌ها تلف خواهند شد. برای جلوگیری از این اتفاق، LWM نخ‌های فعال در سطر بالایی پشته را فشرده می‌کند و وارپ‌هایی برابر با پهنای SIMD ایجاد می‌کند و هر سیکل یکی از آنها را ایشو می‌کند. تفاوت LWM با

[17] Prefetch
[18] Subdivision schemes
[19] Thread-block compaction
[20] Large-warp Microarchitecture

TBC در آن است که LWM نخ‌ها را در تمامی سیکل‌ها همگام می‌کند ولی TBC نخ‌ها را در پایان اجرای بلوک کد همگام می‌کند.

دیاموس و همکاران [۱۲] همگرایی در Thread Frontier را پیشنهاد کرده‌اند تا در نقطه‌ای نزدیک‌تر از پست‌دومیناتور بلافاصله نخ‌های واگرا شده همگرا شوند. این روش مبتنی بر این حقیقت است که همگرایی در نقطه پست‌دومیناتور بلافاصله محافظه‌کارانه است و شرایطی وجود دارد که نخ‌های یک وارپ ممکن است پیش از همگرا شدن، یک بلوک کد را به‌صورت سریال و چندین بار اجرا کنند و این درحالی است که اگر زودتر از پست‌دومیناتور بلافاصله همگرا شده بودند از تمامی اجراهای تکراری جلوگیری می‌شد. ارزیابی‌های آنها نشان می‌دهد که این اتفاق در کنترل جریان‌های بدون‌ساختار [۴۹] به کثرت می‌تواند رخ‌دهد. این روش با اعمال اولویت روی بلوک‌های کد، نخ‌های واگرا شده را سریال و به ترتیبی اجرا می‌کند که اگر در نقطه‌ای پیش از پست‌دومیناتور بلافاصله قابل همگرا شدن هستند، همگرا شوند.

رُو و اِرز [۴۴] روش CAPRI را مبتنی بر TBC معرفی می‌کنند تا از همگام کردن‌های غیرالزامی در انتهای بلوک کد جلوگیری شود. کاپری بر پیشبینی کننده‌ای استوار است که همگام شدن‌های مفید را تشخیص می‌دهد. کاپری با استفاده از یک پیشبینی‌کننده ۲ بیتی اشباعی[21]، نرخ قابل توجهی از همگام کردن‌های غیرالزامی را می‌توان تشخیص دهد. در بین چالش‌های واگرایی انشعاب، CAPRI نرخ فعالیت SIMD و انتظار روی نقطه همگرایی را هدف قرار می‌دهد.

برونی و همکاران [۴] نرخ فعالیت SIMD را هدف قرار می‌دهند و مکانیزم‌های SBI، SWI، و SBI-SWI را برای در میان کردن دستورالعمل‌ها معرفی می‌کنند که در آنها دستورالعمل دومی برای پرکردن خطوط غیرفعال دستور اصلی ایشو می‌شود. تفاوت این روش‌ها در نحوه‌ی انتخاب دستور دوم است. دستور دوم در SBI از مسیرهای واگرایی پایین پشته‌ی همان وارپ انتخاب می‌شود، دستور دوم در SWI از مسیر واگرایی فعال وارپ‌های دیگر آورده می‌شود، و SBI-SWI ترکیب هر دو روش SBI و SWI است و می‌تواند دستور دوم را از مسیرهای واگرایی غیرفعال همان وارپ یا مسیر واگرایی فعال وارپ‌های دیگر ایشو کند. در حین اجرا، هر خط از SIMD با یک ماسک عملیاتی پیکربندی پویا می‌شود که یکی از این دو دستور ایشو شده را اجرا کند.

[21] 2-bit saturating counter

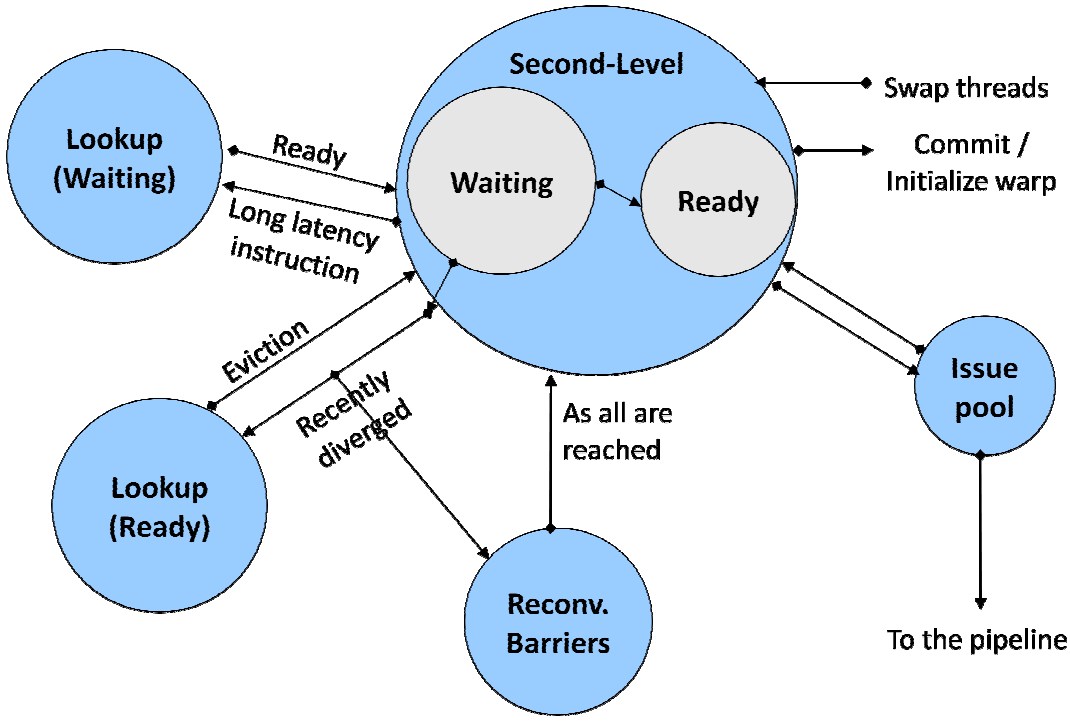
فصل ۴

روش پیشنهادی CROWN

روش پیشنهادی CROWN را به عنوان راه حلی جامع برای کاهش هر سه چالش واگرایی انشعاب، که در فصل ۲-۵-۱ بیان شد، ارائه می‌کنیم. در این روش برای «بهبود نرخ فعالیت SIMD» (چالش اول)، نخ‌های واگرا شده در وارپ‌های جدیدی گروه‌بندی می‌شوند. افزون بر این، از مکانیزم متفاوتی برای همگرایی نخ‌ها در نقطه بازهمگرایی استفاده شده است. نخ‌های واگرا شده در مسیرهای واگرایی متفاوت همروند اجرا می‌شوند تا «اجرای متوالی مسیرهای واگرایی» (چالش دوم) کاملا برطرف شود. گروه‌بندی نخ‌های واگراشده از وارپ‌های متفاوت در یک وارپ، «انتظار روی نقطه بازهمگرایی» (چالش سوم) را کاهش می‌دهد زیرا در این وضعیت نخ‌ها در یک وارپ اجرا می‌شوند و اجرای آنها همزمان انجام می‌شود (بصورت وارپ‌هایی سریال اجرا نخواهند شد) و سریع‌تر به نقطه بازهمگرایی می‌رسند. افزون بر این، برای کاهش بیشتر انتظار روی نقطه بازهمگرایی (چالش سوم)، نخ‌ها در وارپ‌های کوچکتری (برابر با پهنای SIMD) گروه‌بندی می‌شوند. همانطوری که پیش‌تر بیان شد، وارپ‌های کوچکتر واگرایی حافظه و واگرایی انشعاب کمتری دارند و نخ‌ها در گروه‌های کوچکتری سنکرون می‌شوند که باعث می‌شود سریع‌تر به نقطه همگرایی برسند. در ادامه این فصل روش پیشنهادی CROWN برای نایل شدن به موارد فوق را معرفی می‌کنیم. ابتدا عملکرد سطح بالایی از CROWN را شرح می‌دهیم. سپس معماری پیشنهادی برای پیاده‌سازی CROWN را شرح می‌دهیم و در پایان سربار سخت‌افزاری پیاده‌سازی آن را ارزیابی می‌کنیم.

۴-۱ نگاه سطح بالا به عملکرد CROWN

در CROWN، مادامی که واگرایی انشعاب رخ نداده است، عملکرد آن مشابه روش پایه SBR است. هنگامی که واگرایی انشعاب رخ دهد، وارپِ واگراشده بی‌اعتبار می‌شود و دو وارپ جدید ایجاد می‌شوند. همچنین سنکرون کننده‌ای برای همگرا کردن نخ‌ها در نقطه بازهمگرایی رزرو می‌شود. وارپ‌های جدید مسیرهای واگرایی را اجرا می‌کنند و اجرای آنها درمیان می‌شود. هنگام رسیدن نخ‌ها به نقطه بازهمگرایی،



شکل ۴-۱: ماشین حالت انتقال وضعیت نخ‌ها در طول اجرا در CROWN. هر وضعیت در این ماشین به یک جدول در معماری نگاشت می‌شود که نگهدارنده‌ی تعدادی نخ در دانه‌بندی وارپ است.

وارپ نخ‌ها بی‌اعتبار می‌شود و سنکرون کننده رسیدن نخ‌ها را علامت گذاری می‌کند. زمانی که تمامی نخ‌هایی که سنکرون کننده منتظرشان هست، علامت گذاری شوند تمامی نخ‌ها به نقطه بازهمگرایی رسیده‌اند. در این حالت سنکرون کننده وارپ جدیدی متشکل از نخ‌های سنکرون شده به سیستم اضافه می‌کند و اجرای نخ‌ها از نقطه بازهمگرایی ادامه پیدا می‌کند.

وارپ‌های ساخته شده پس از واگرایی به طور موقت در جدولی (جدول گروه‌بندی پویا) نگهداری می‌شود تا در صورت امکان با دیگر وارپ‌های واگرا شده ترکیب شوند و نرخ فعالیت SIMD بهبود یابد. این وارپ‌ها موقت از لیست زمان‌بندی و اجرای وارپ‌ها خارج می‌شوند و زمانی که لیست زمان‌بندی رو به خالی شدن باشد، از این وارپ‌ها برای پر کردن دوباره لیست استفاده می‌کند و در این زمان این وارپ‌ها از جدول گروه‌بندی پویا خارج می‌شوند.

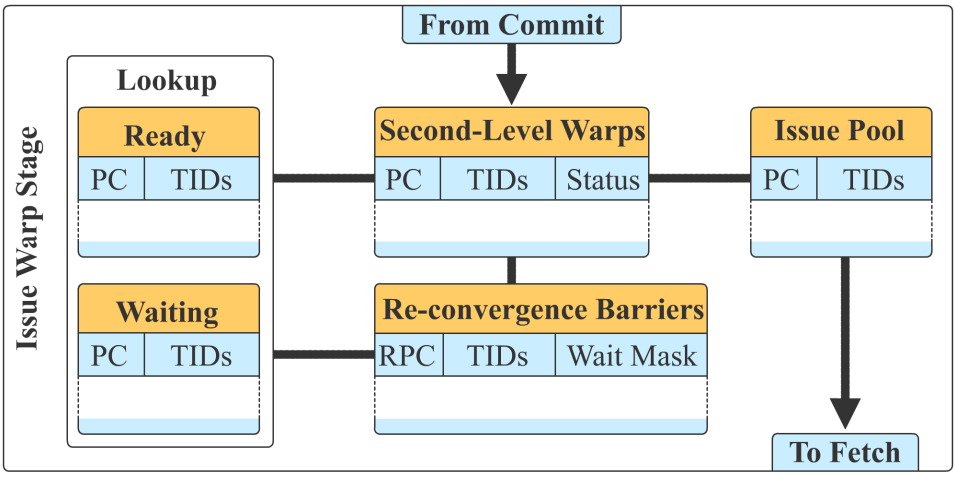
۴-۲ ماشین حالت تغییر وضعیت نخ‌ها و معماری

۴-۲-۱ ماشین حالت تغییر وضعیت نخ‌ها

پیش از آنکه معماری پیشنهادی برای پیاده‌سازی مکانیزم CROWN را معرفی کنیم، ماشین حالت شکل ۴-۱ را ارائه می‌کنیم که انتقال بین وضعیت‌هایی که هر نخی در طول اجرایش ممکن است تجربه کنند را نشان می‌دهد. تعدادی از وارپ‌های آماده برای اجرا در سطح اول زمان‌بندی (issue pool) قرار می‌گیرند و وارپِ بعدی برای اجرا از بین این وارپ‌ها انتخاب می‌شود. هنگامی که وارپ آماده‌ای برای اجرا انتخاب شد، از سطح اول خارج می‌شود و به سطح دوم قرار داده می‌شود و به عنوان وارپ منتظر علامت‌زده می‌شود. پس از اینکه اجرای دستورِ وارپِ منتظر تمام شد، هنگام commit کردن وارپ، وضعیت آن در سطح دوم به آماده تغییر پیدا می‌کند و هنگامی که تعداد وارپ‌های موجود در سطح اول رو به اتمام باشد، وارپ آماده از سطح دوم به سطح اول آورده می‌شود. توجه شود که در ابتدا نیز، وقتی بلوک نخ جدیدی به هسته سپرده می‌شود، وارپ‌های آن بلوک نخ در سطح دوم قرار می‌گیرند و در زمان مقتضی وارد سطح اول می‌شوند.

هنگام commit کردن دستور یک وارپ اگر واگرایی انشعاب رخ داده باشد، این وارپ منتظر در سطح دوم باطل می‌شود و حداکثر دو وارپ آماده شامل نخ‌های هر مسیر واگرایی ایجاد می‌شوند. این دو وارپ موقتا در جدول گروه‌بندی پویای آماده (Ready Lookup) قرار می‌گیرند تا در صورت امکان با دیگر وارپ‌های واگرا شده (موجود در همین جدول) ترکیب شوند تا نرخ فعالیت SIMD بهبود یابد. امکان‌پذیر است که وارپ‌های موجود در این جدول در هر زمانی به سطح دوم منتقل شوند و سپس با انتقال به سطح اول اجرا شوند. ما این انتقال را زمانی انجام می‌دهیم که در سطح دوم کمتر از ۲ وارپ آماده وجود داشته باشد.

همزمان با ایجاد وارپ‌های آماده‌ای در هر مسیرهای واگرایی، سدّی برای سنکرون کردن نخ‌های وارپِ واگراشده در نقطه بازهمگرایی در جدول سدهای بازهمگرایی (Re-convergence barriers) رزرو می‌شود. این سد شناسه نخ‌ها و آدرس نقطه بازهمگرایی را علامت گذاری می‌کند. هنگامی که نخی به این نقطه برسد از ادامه اجرای آن جلوگیری می‌شود و رسیدن این نخ، در سدِّ متناظر علامت‌گذاری می‌شود. هنگامی که تمامی نخ‌هایی که سد منتظر آنها است به نقطه بازهمگرایی برسند، سد می‌تواند به یک وارپ تبدیل شود



شکل ۴-۲: معماری پیشنهادی برای پیاده‌سازی CROWN.

و اجرا از نقطه بازهمگرایی ادامه پیدا کند. سدهای تبدیل شده به وارپ زمانی که سطح دوم خالی از وارپ آماده باشد به سطح دوم انتقال پیدا می‌کنند و سپس می‌توانند به سطح اول منتقل و اجرا شوند.

مدت زمانی که یک وارپ در جدول گروه‌بندی پویای آماده باقی می‌ماند با احتمال ترکیب آن با وارپ‌های واگراشده‌ی دیگر رابطه مستقیم دارد. هرچه یک وارپ بیشتر در این جدول بماند، محتمل‌تر خواهد بود که با وارپ واگرا شده‌ی دیگری ترکیب شود. بر خلاف این مطلوب، مشاهدات نشان می‌دهد که وارپ‌ها به مدت کوتاهی در این جدول می‌مانند زیرا هنگامی که تعداد وارپ‌های آماده در سطح دوم کم شود (به علت تاخیر بزرگ دسترسی به حافظه یا سنکرون کننده‌های درون بلوک نخی این اتفاق متداولا رخ می‌دهد)، به ناچار تمامی وارپ‌های موجود در این جدول رفته رفته به سطح دوم می‌روند تا تاخیر اجرا وارپ‌های منتظر را مخفی کنند. با توجه به این مشاهده، برای اینکه احتمال ترکیب وارپ‌های واگرا شده بالارود، گروه‌بندی پویای وارپ‌های منتظر (Waiting Lookup) را پیشنهاد می‌کنیم. این گروه‌بندی مبتنی بر این مشاهده کلیدی است که اجرای دستورالعمل‌های دسترسی به حافظه سراسری معمولا تاخیر زیادی دارند و از این تاخیر می‌توان به عنوان یک سنکرون کننده برای ترکیب وارپ‌های واگراشده استفاده کرد. بدین ترتیب، هنگامی که وارپی دستوری با تاخیر زیاد را اجرا کرد، به جدول گروه‌بندی پویا وارپ‌های منتظر (Waiting Lookup) منتقل می‌شود و وارپ‌های دیگری که این دستور را اجرا کنند نیز به این جدول منتقل می‌شود و در صورت امکان با وارپ‌ها موجود در این جدول ترکیب می‌شود. وارپ‌های این جدول

به محض اینکه به وضعیت آماده درآیند به سطح دوم منتقل می‌شود. همانطوری که در بخش ۶-۳-۱ ارزیابی خواهد شد، این جدول در برخی موارد برای بهبود نرخ فعالیت SIMD به میزان قابل توجهی تاثیرگذار عمل می‌کند.

۴-۲-۲ معماری

در شکل ۴-۲ معماری پیشنهادی برای پیاده‌سازی CROWN ترسیم شده است که متناظر با ماشین حالت شکل ۴-۱ است. زمان‌بندی وارپ‌های آماده برای اجرا در issue pool انجام می‌شود که جدولی با ظرفیت بسیار کمتری (۸ وارپ) نسبت به جدول وارپ‌های سطح دوم است. هر سطر در این جدول به یک وارپ اختصاص دارد و شمارنده برنامه و شناسه نخ‌های آن وارپ را نگهداری می‌کند.

وارپ‌های سطح دوم می‌توانند آماده یا منتظر (منتظر اتمام دستورالعمل خود یا منتظر روی سدّ سنکرون کننده‌ی درون بلوک نخی) باشند و هنگام commit وضعیت وارپ‌ها در این جدول بروز می‌شود. وارپ‌های جداول دیگر پیش از انتقال به issue pool باید به سطح دوم منتقل شوند و این جدول، واسط بین جداول دیگر و issue pool است. در هنگامی که به علت واگرایی های متداول تعداد وارپ‌ها این جدول بیش از سطرهای جدول شود، تعدادی از وارپ‌ها می‌توانند به خارج از هسته منتقل شوند.

جدول‌های گروه‌بندی پویا (آماده یا منتظر) تعداد کمی (۴ تا ۸) سطر دارند و fully-associative هستند. تمامی سطرهای این جداول برای یافتن وارپی برای ترکیب شدن با نخ واگراشده جستجو می‌شود و وارپ هدف باید ۱) شمارنده برنامه‌ای یکسان با نخ واگراشده داشته باشد و ۲) خط متناظر با نخ واگراشده، در وارپ هدف خالی باشد (نخی در آن خط از وارپ هدف نباشد).

جدول نگهداری سدهای بازهمگرایی جدولی 8-way set-associative است و می‌تواند تعداد زیادی سطر داشته باشد. هر مجموعه از این جدول، تعدادی سنکرون کننده (به اندازه‌ی تعداد راه‌های جدول) برای دسته‌ای از نخ‌ها نگه می‌دارد. مجموعه‌ای که نخ به آن نگاشت می‌شود از طریق بیت‌های پرارزش شناسه‌ی نخ معین می‌شود.

اندازه جدول‌های گروه‌بندی پویای وارپ‌های آماده (Ready Lookup)، گروه‌بندی پویای وارپ‌های منتظر (Waiting Lookup)، و سدهای بازهمگرایی (re-convergence barriers) از مهمترین پارامترهای

جدول ۴-۱: سربار سخت‌افزاری پیاده‌سازی CROWN در تکنولوژی ساخت ۹۰ نانومتری برای هر هسته. تعداد بیت‌های Tag در ماجول‌ها حاصل جمع بیت‌های مقایسه برای شمارنده برنامه (۳۲ بیت) و شناسه ۸ نخ در وارپ است. ۷ بیت (۳-۱۰ بیت) برای شناسایی منحصر به فرد هریک از ۱۰۲۴ نخ همروند کافی است زیرا خط نخ در SIMD زمان گروه‌بندی تغییر نمی‌کند و نیازی به ذخیره ۳ بیت کم‌ارزش نیست.

	Module Specification					Area (mm^2)	
	#entries	#sets	Associativity	Tag bits	Row Size (bits)	Tag	Data
Reconv. Barrier	128	16	8	32+(8*(10-3))	32+(8*(10-3))+8	0.035	0.137
Second level	512	512	1		32+(8*(10-3))+2		0.123
Lookup Ready	8	1	8	32+(8*(10-3))	32+(8*(10-3))	0.020	0.091
Lookup Waiting	8	1	8	32+(8*(10-3))	32+(8*(10-3))	0.020	0.091
Total area						0.5185 mm^2	

قابل پیکربندی CROWN برای مصالحه هزینه/کارایی هستند. در بخش بعدی سربار بزرگترین پیکربندی ارزیابی شده‌ی CROWN در این پژوهش را محاسبه می‌کنیم و در بخش ۶-۳-۱ حساسیت بروندهی به اندازه این جداول را ارزیابی خواهد شد.

۴-۳ ارزیابی سربار سخت‌افزاری

سربار سخت‌افزاری روش پیشنهادی CROWN را به دو بخش تقسیم می‌کنیم و هریک را جداگانه محاسبه می‌کنیم: ۱) جدول‌های پیاده‌سازی CROWN و ۲) دیکود کردن چندین آدرس در رجیسترفایل.

۴-۳-۱ جدول‌های پیاده‌سازی CROWN

سربار سخت‌افزاری طراحی پیشنهادی برای پیاده‌سازی CROWN به پارامترهای پیکربندی آن (از قبیل تعداد سطرهای موجود در هریک از جدول‌های سطح دوم زمان‌بندی، سدهای بازهمگرایی، گروه‌بندی پویای آماده، و گروه‌بندی پویای منتظر) بستگی دارد. در این بخش با فرض ۵۱۲ سطر در جدول زمان‌بندی سطح دوم، ۱۲۸ سطر در جدول سدهای بازهمگرایی، ۸ سطر در جدول گروه‌بندی پویای آماده، و ۸ سطر در جدول گروه‌بندی پویای منتظر، تخمینی از سربار سخت‌افزاری CROWN بدست می‌آوریم. در بخش ۶-۳-۱ تاثیر کمتر شدن اندازه جداول بر کارایی CROWN ارزیابی می‌شود. جدول ۴-۱ مساحت جدول‌های مختلف CROWN را گزارش می‌کند که بوسیله CACTI 6.5 [۵۲] در تکنولوژی ۹۰ نانومتری بدست آمده است. همانطوری که گزارش شده است، مساحت تخمینی برای پیاده‌سازی CROWN در

تکنولوژی ۹۰ نانومتر برابر ۰.۵۲ میلی‌متر مربع برای هر هسته است. این سربار برای ۱۶ هسته برابر ۸.۳ میلی‌متر مربع خواهد شد. معماری مشابه Tesla در ۹۰ نانومتر بر تراشه‌ی ۴۷۰ میلی‌متر مربعی ساخته شده است [۳۴]. می‌توان نتیجه گرفت که پیاده‌سازی CROWN برای معماری‌ای شبیه به Tesla، ۱.۷۷٪ سربار سخت‌افزاری خواهد داشت.

۴-۳-۲ دیکودکردن چندین آدرس در رجیسترفایل

در طراحی پایه فرض می‌شود که هر ردیف از رجیسترفایل ۱۲۸ بایت (۳۲ ثباتِ ۴ بایتی) است. عملوند نخ‌های یک وارپ در یک ردیف از رجیسترفایل ذخیره می‌شوند و هنگام خواندن عملوند، با دیکود کردن یک آدرس و دسترسی/واکشی یک سطر می‌توان یک عملوند از دستور را برای تمامی نخ‌های وارپ خواند. اما در روش پیشنهادی CROWN هنگامی که نخ‌هایی از وارپ‌های متفاوت در قالب یک وارپ ایشو می‌شوند، عملوند نخ‌های این وارپ در سطرهای متفاوتی قرار خواهند داشت و با واکشی یک ردیف نمی‌توان کلیه عملوندهای این نخ‌ها را واکشی کرد. تحقیقات پیشین [۱۳، ۱۴، ۳۸] استفاده از چندین دیکودکننده آدرس برای بانک‌های مختلف رجیسترفایل را پیشنهاد کرده‌اند. ارزیابی آنها نشان می‌دهد که این طراحی مساحت رجیسترفایل را ۱۸.۷٪ افزایش می‌دهد که برابر ۲.۵٪ از مساحت تراشه GPU خواهد بود.

فصل ۵

روش پیشنهادی DWR

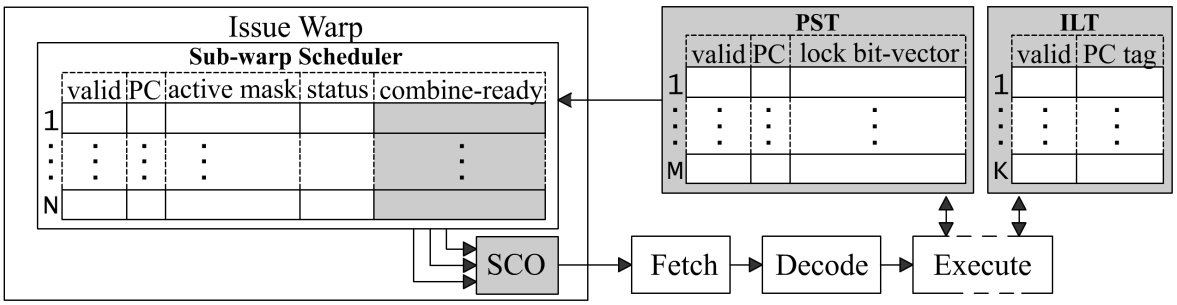
هدف DWR نایل شدن به مزایای هردو طراحی وارپ بزرگ و وارپ کوچک است. DWR تکنیکی در لایه معماری است که با وارپ کوچک شروع (برابر با پهنای SIMD) می‌کند و با توجه به رفتار کاربرد، برای استفاده از وارپ بزرگ آموزش می‌بیند. اندازه وارپ‌های کوچک برابر پهنای SIMD خواهند بود و از این به بعد به آنها زیروارپ[1] نیز می‌گوییم. با درمیان کردن اجرای وارپ‌های کوچک، واگرایی حافظه و واگرایی انشعاب کم می‌شود. DWR با افزایش اندازه وارپ به‌صورت پویا، الحاق دسترسی به حافظه را بالا می‌برد که این افزایش اندازه متکی بر استفاده از همگام‌کننده‌ها برای همگام کردن و تجمیع کردن چندین زیروارپ است. DWR زیروارپ‌ها را مستقل از هم زمان‌بندی می‌کند و آنها را برای اجرای دستورالعمل‌های حافظه همگام می‌کند تا در قالب یک وارپ بزرگ این دستورالعمل‌ها را اجرا کنند. برای پیاده‌سازی این همگام‌کننده، زمان‌بند وارپ‌ها و مجموعه دستورالعمل‌های ماشین (ISA [2]) توسعه پیدا می‌کنند. معماری DWR در شکل ۱.۵ نشان داده شده‌است.

در بخش اول، معماری پایه‌ی پیشنهادی معرفی می‌شود. رهایی از بن‌بست و جلوگیری از همگام شدن‌های غیرالزامی به‌ترتیب در بخش‌های دوم و سوم ارائه می‌شوند. عملکرد دستورالعمل اضافه شده برای رهایی از بن‌بست و جلوگیری از همگام شدن‌های غیرالزامی را در بخش چهارم شرح می‌دهیم. نمونه‌ای از عملکرد DWR در بخش چهارم آمده است. در انتها، سربار سخت‌افزاری تحمیل شده را ارزیابی می‌کنیم.

۵-۱ معماری

DWR، وارپ‌هایی با اندازه‌های متفاوت گروه‌بندی و ایشو می‌کند؛ وارپ‌های بزرگ برای دستورالعمل‌های ویژه‌ای استفاده می‌شوند و ماباقی دستورالعمل‌ها با زیروارپ‌ها اجرا می‌شوند. دسته‌ای از زیروارپ‌ها که

[1] Sub-warp
[2] Instruction Set Architecture



شکل ۵-۱: معماری DWR. تفاوت DWR با معماری پایه با رنگ خاکستری مشخص شده است. زمان‌بند N زیروارپ مستقل را زمان‌بندی می‌کند. دستور همگام کننده از PST و ILT برای همگام کردن زیروارپ‌ها استفاده می‌کند. برای هر یک از M وارپ بزرگ، یک سطر در PST وجود دارد تا زیروارپ‌های آن را همگام کند. سطرهای ILT برای چشم‌پوشی کردن برخی از سدهای حساس در هنگام همگام کردن استفاده می‌شود. SCO زمانی که زیروارپ‌ها همگام شوند یک وارپ بزرگ ایشو می‌کند.

«زیروارپ‌های همکار» نامیده می‌شوند، همگام شده و یک وارپ بزرگ برای اجرای یک دستورالعمل ویژه تشکیل می‌دهند. دستورالعمل‌های ویژه شامل مجموعه‌ای ثابت (و استاتیک) از دستورالعمل‌های سطح‌پایین PTX هستند که آنها را دستورالعمل‌های «حساس به وارپ بزرگ» یا به اختصار «حساس» می‌خوانیم. تفاوت اصلی دستورالعمل‌های حساس با دیگر دستورالعمل‌ها این است که دستورالعمل‌های حساس با وارپ بزرگ سریع‌تر اجرا می‌شوند. دستورالعمل‌های غیرحساس همیشه با زیروارپ اجرا می‌شوند. در طرف مقابل، دستورالعمل‌های حساس، با وارپ‌های بزرگ اجرا می‌شوند که متشکل از چندین زیروارپ است. زمان‌بند وارپ در DWR، هنگامی که تشخیص دهد زیروارپ‌های همکار همگام شده‌اند و آماده‌ی اجرا هستند، آنها را ترکیب می‌کند و یک وارپ بزرگ ایشو می‌کند. برای گرفتن این تصمیم، یک بیت برای هر زیروارپ (که به آن بیت تعیین وضعیت ترکیب-آماده می‌گوییم) استفاده می‌شود.

از آنجایی که زمان‌بند وارپ‌ها آزاد است که زیروارپ‌ها را در هر ترتیبی انتخاب کند، برخی از زیروارپ‌ها ممکن است زودتر از دیگران به دستورالعمل‌های حساس برسند. برای تضمین حضور تمامی زیروارپ‌ها در هنگام اجرا دستور حساس، همگام‌کننده‌ای قبل از دستور حساس تحمیل می‌شود. این همگام کننده می‌تواند به‌صورت ایستا یا پویا تحقق پیدا کند. در *روش ایستا*، که در این پژوهش از آن استفاده می‌کنیم، ISA و سخت‌افزار برای پشتیبانی از همگام‌کننده بین زیروارپی توسعه پیدا می‌کنند. در زمان کامپایل، هر دستور حساس با دو دستور جایگزین می‌شود: اول همگام‌کننده‌ی بین زیروارپ‌های همکار، و

الگوریتم ۵-۱: الف) دنباله دستورالعمل‌های PTX اصلی برای معماری پایه. ب) کد تولید شده برای DWR که از سنکردن کردن بین زیروارپی روی دستورالعمل‌های حساس پشتیبانی می‌کند.

```
cvt.u64.s32     %rd1, %r3;
ld.param.u64    %rd2, [__parm1];
add.u64         %rd3, %rd2, %rd1;
-----------------------------------
ld.global.s8    %r5, [%rd3+0];
-----------------------------------
mov.u32         %r6, 0;
setp.eq.s32     %p2, %r5, %r6;
@%p2 bra        $Lt_0_5122;
mov.s16         %rh2, 0;
-----------------------------------
st.global.s8    [%rd3+0], %rh2;
-----------------------------------
```

(الف)

```
cvt.u64.s32     %rd1, %r3;
ld.param.u64    %rd2, [__parm1];
add.u64         %rd3, %rd2, %rd1;
-----------------------------------
bar.synch_partner       0;
ld.global.s8    %r5, [%rd3+0];
-----------------------------------
mov.u32         %r6, 0;
setp.eq.s32     %p2, %r5, %r6;
@%p2 bra        $Lt_0_5122;
mov.s16         %rh2, 0;
-----------------------------------
bar.synch_partner       0;
st.global.s8    [%rd3+0], %rh2;
-----------------------------------
```

(ب)

دوم خود دستور حساس. دستور اول، که به آن همگام‌کننده‌ی حساس یا سدّ[3] حساس می‌گوییم، تضمین می‌کند که تمامی زیروارپ‌ها به این نقطه رسیده‌اند. دستور دوم با یک وارپ بزرگ اجرا می‌شود. الگوریتم ۵-۱(الف) بخشی از کد کرنل (مرتبط با محک BFS) در نحو PTX [۴۲] را نشان می‌دهد. الگوریتم ۵-۱(ب) کد تبدیل شده را نشان می‌دهد که برای DWR کامپایل شده است. در این الگوریتم، bar.synch_partner دستورالعمل جدیدی است که همگام‌کننده‌ی بین زیروارپ‌ها است. در صورتی که روش ایستا مطلوب نباشد، می‌توان از *روش پویا* استفاده کرد که ابتدا دستور حساس پس از اولین بار دیکود شدن تشخیص داده می‌شود (و به لیستی اضافه می‌شود) و زیروارپ‌ها را از آن به بعد روی این دستور همگام می‌کند. روش پویا، باینری‌های DWR را سازگار با طراحی پایه نگه می‌دارد اما نیاز به فاز آموزش دیدن و یادگیری قبل از تشخیص دستورالعمل‌های حساس دارد.

انتخاب دستورالعمل‌های حساس. با استفاده از واژگان مجموعه دستورالعمل‌های مجازی PTX، کاندیداهای دستورالعمل‌های حساس می‌توانند ذخیره/بارگذاری در/از فضای سراسری/محلی یا بارگذاری از فضای ثابت باشند. این دستورالعمل‌ها تنها دستورالعمل‌هایی هستند که به وضوح به حافظه سراسری دسترسی پیدا می‌کنند. معماری پایه مطالعه شده در این پژوهش، از الحاق دستورالعمل‌های دسترسی به

[3] Barrier

فضای ثابت پشتیبانی نمی‌کند. در نتیجه، ما دستورالعمل‌های ذخیره/بارگذاری در/از فضای سراسری/محلی را به عنوان دستورالعمل‌های حساس در نظر می‌گیریم.

ترکیب کننده زیروارپ‌ها. واحد ترکیب کننده زیروارپ‌ها[4] یا (SCO) برای تشکیل وارپ‌های بزرگ در هنگام ایشو کردن دستورالعمل‌های حساس استفاده می‌شود. زیروارپی که دستور سدّ حساس را اجرا کرده از وضعیت آماده خارج می‌شود و منتظر می‌ماند تا همگام‌کننده وضعیت آن را به آماده–ترکیب علامت بزند. هنگامی که بیت آماده–ترکیب برای زیروارپی فعال شده باشد، نشان می‌دهد که تمام زیروارپ‌های همکار آن به دستور همگام‌کننده حساس رسیده‌اند و آماده هستند که ترکیب‌شوند و دستور حساس مرتبط را اجرا کنند. هنگامی که یکی از این زیروارپ‌ها برای زمان‌بندی انتخاب شود، دیگر زیروارپ‌های همکار نیز ایشو می‌شوند. سپس SCO ماسک فعالیت زیروارپ‌های آماده–ترکیب را تجمیع می‌کند و یک وارپ بزرگ ایشو می‌کند. بیشینه تعداد زیروارپ‌های قابل الحاق (اندازه بزرگترین وارپ) از پارامترهای قابل پیکربندی ایستا در DWR است. هرچه این بیشینه بزرگتر باشد، موقعیت بیشتری برای الحاق دسترسی به حافظه بین وارپ‌ها ایجاد می‌شود ولی از طرف دیگر سربار همگام کردن بیشتری تحمیل می‌کند. در این پژوهش، بیشینه اندازه وارپ بزرگ را ۲ برابر، ۴ برابر، و ۸ برابر اندازه زیروارپ در نظر می‌گیریم و ارزیابی می‌کنیم.

۵-۲ رهایی از بن‌بست

معماری معرفی شده در بالا ممکن است در یکی از دو وضعیت زیر به بن‌بست‌ای منتهی شود وارپ‌ها در وضعیت منتظر روی سنکرون کننده باقی بمانند:

۱) سدّ حساس به‌علاوه سدّ حساس دیگر

۲) سدّ حساس به‌علاوه __syncthreads()

عموما، در هر دو مورد فوق، زیروارپ‌های همکار منتظر روی دو سدّ متفاوت می‌مانند که از رهایی یکپارچه آنها جلوگیری می‌کند. این اتفاق وقتی می‌افتد که واگرایی انشعاب درون یک وارپ بزرگ رخ دهد و زیروارپ‌ها مسیرهای واگرایی متفاوت بگیرند و هریک دستورالعمل‌های حساس متفاوتی اجرا کنند. الگوریتم ۵-۲، دو مثال سطح بالای شبیه به کودا ارائه می‌کند تا رخ‌داد بن‌بست را واضح سازد.

[4] Sub-warp Combiner

الگوریتم ۵-۲: وضعیت بن‌بست که در DWR پایه رخ می‌دهد. (الف) یکی از زیروارپ‌ها روی سدّ حساس #۲ منتظر می‌ماند و دیگری روی سدّ حساس #۴ منتظر می‌ماند. (ب) یکی از زیروارپ‌ها روی سدّ حساس #۲ و زیروارپ دیگر روی syncthreads در دستور 4# منتظر می‌ماند.

```
1: if( sub_warp_id == 0){
2:       regA = gmem[idxA];
3: }
4: regB = gmem[idxB];
```

(الف)

```
1: if( sub_warp_id == 0){
2:       regA = gmem[idx];
3: }
4: __syncthreads();
```

(ب)

الگوریتم ۵-۳: وضعیتی در API استاندارد کودا که انتظار می‌رود به بن‌بست منتهی شود.

```
if( warp_id == 0){
__syncthreads(); // warp-0 is locked here
}else{
__syncthreads(); // warp-1 is locked here
}
```

این موارد بن‌بست شبیه به آنچیزی است که می‌تواند در API استاندارد کودا رخ‌دهد (الگوریتم ۵-۳). اگرچه، همانطوری که توسط وُنگ و همکاران [۴۸] شرح داده شده است، این وضعیت به بن‌بست منتهی نمی‌شود زیرا سخت‌افزار همگام کننده، نخ‌ها را روی دستورالعمل مشخصی همگام نمی‌کند و فقط نخ‌ها را قفل می‌کند تا زمانی که همه‌ی آنها ۱) به دستوری از نوع synchthreads()__ یا ۲) به دستور خروج برسند. درنتیجه، در هر دو وضعیت ارائه شده در الگوریتم ۵-۲، بن‌بست با رها کردن هردوی زیروارپ‌ها می‌تواند پیشگیری می‌شود. توجه شود که این زیروارپ‌ها نمی‌توانند یک وارپ بزرگ مشترک تشکیل دهند چون PCهای متفاوتی دارند. در نتیجه، در این وضعیت، زیروارپ‌های همکار در وارپ‌های متفاوتی گروه‌بندی می‌شوند.

۵-۳ همگام‌کردن انتخابی

همگام کردن زیروارپ‌های همکار در وضعیت‌های مشابه الگوریتم ۵-۲، منفعت الحاق کمی دارد درحالی که همگام کردن سربار زیادی برای گذردهی دارد. به این نوع همگام کردن‌های کم‌منفعت برای گذردهی، سدّ حساس بیهوده می‌گوییم. سدّ حساس‌های بیهوده در برنامه‌هایی رایج است که مرتبه‌ی بالایی از واگرایی انشعاب در آنها رخ می‌دهد. مشاهدات ما نشان می‌دهد که محک‌هایی از جمله BFS، MU، MP، و NQU در این دسته قرار می‌گیرند. تشخیص سدّ حساس بیهوده به صورت ایستا امکان‌پذیر نیست زیرا سدّ حساس بیهوده به شرایط کنترلی اجرای هر نخ بستگی دارد که اغلب پدیده‌ای پویا است [۵۱]. DWR این سدها را

با استفاده از دستور bar.synch_partner و به‌صورتی که در ادامه شرح داده شده است تشخیص می‌دهد. زمانی که دستور bar.synch_partner تشخیص دهد که زیروارپ‌ها از شمارنده برنامه‌های متفاوتی همگام شده‌اند، یکی از PCها را در جدولی ذخیره می‌کند. این جدول، جدول لیست چشم‌پوشی[5] یا ILT نامیده می‌شود و در شکل ۵-۱ نشان داده شده است. لیست چشم‌پوشی به‌صورت پویا در هنگام اجرای گرید روی هر هسته بروز رسانی می‌شود و PCهای سدهای حساس بیهوده را ذخیره می‌کند. اطلاعات این جدول تنها توسط دستور bar.synch_partner بروز رسانی می‌شود و تنها این دستور لازم است این اطلاعات را بارگذاری کند. این جدول با کاهش همگام کردن‌های بیهوده، نقش مهمی در بهبود گذردهی دارد بدین‌ترتیب که زیروارپ روی سدّ حساس قفل نمی‌شود اگر شمارنده‌ی برنامه‌ی آن در لیست چشم‌پوشی ثبت شده باشد.

۵-۴ دستور سدّ حساس

در این بخش عملکرد دستور bar.synch_partner را شرح می‌دهیم. پیشتر گفته شد که گروهی از زیروارپ‌ها در DWR که به‌صورت ایستا پیکربندی‌شده‌اند تا روی دستورالعمل‌های حساس با هم همگام شوند، زیروارپ‌های همکار نام دارند. برای مدیریت همگام کردن هر گروه زیروارپ‌های همکار، یک سطر برای هر گروه در جدول همگامی همکارها[6] یا PST ذخیره می‌شود. همانطوری که در شکل ۵-۱ نشان داده شده است، هر سطر PST شامل شمارنده برنامه و بردار بیتی قفل است. دستور bar.synch_partner روی دو ورودی عمل می‌کند؛ شناسه زیروارپ و شمارنده برنامه‌ی دستور. هنگام اجرای دستور bar.synch_partner، اگر PC در ILT وجود داشت، عملیات اضافی دیگری انجام نمی‌شود. در غیر این صورت، عملیات‌های زیر به‌صورت سریال در دو مرحله و در طول اجرای دستور انجام می‌شوند:

مرحله اول: بروز کردن PC، بردار بیتی قفل، و ILT. اگر سطر گروه همکار معتبر نیست، PC سطر بروز می‌شود و بیت مرتبط زیروارپ در بردار بیتی قفل ’1‘ می گردد. اگر سطر گروه معتبر است و PC موجود برابر با PC زیروارپ است، تنها بردار بیتی بروز می‌شود. اگر سطر گروه معتبر است و PC برابر با PC دستورِ سد نیست، بردار بیتی بروز می‌شود و PC زیروارپ به ILT اضافه می‌شود تا این دستور در همگام کردن‌ها آینده چشم‌پوشی شود.

[5] Ignore list table
[6] Partner-synch table

مرحله دوم: بروز کردن وضعیت زیروارپ‌ها. اگر بردار بیتی تماما '1' است، سد تمام زیروارپ‌های همکار را آزاد می‌کند و آنها را با عنوان ترکیب-آماده در زمان‌بند وارپ‌ها علامت می‌زند. در غیر این صورت، زیروارپ علامت «منتظر» می‌خورد و منتظر دیگر زیروارپ‌ها می‌ماند.

توجه شود که در هنگامی که وارپ بزرگ به علت کمبود تعداد نخ‌های همروند تعداد کمتری زیروارپ فعال نسبت به اندازه بردار بیتی دارد، بردار بیتی هیچ‌وقت تماما '1' نمی‌شود. برای اینکه مشخص شود تمامی زیروارپ‌های موجود وارپ بزرگ به سدّ حساس رسیده‌اند از عبارت منطقی زیر استفاده کرده‌ایم:

$$AND_{\mathrm{m=X}}^{\mathrm{Y}}(\overline{\mathrm{A[n * W + m]\ AND\ \overline{B[m]}}}) \qquad (۱-۵)$$

که در آن n شناسه وارپ بزرگ، W اندازه وارپ بزرگ بر حسب تعداد زیروارپ‌ها، X شناسه اولین زیروارپِ وارپِ بزرگ، Y شناسه آخرین زیروارپِ وارپِ بزرگ است، B بردار بیتی قفل متناظر با وارپ بزرگ n ام است که از جدول PST وارد شده، و A بردار بیتی اعتبار زیروارپ‌ها است که از زمان‌بند زیروارپ‌ها وارد می‌شود. عبارت منطقی فوق مادامی که زیروارپی از وارپ بزرگ n ام در زمان‌بند وارپ‌ها معتبر است و هنوز به سدّ حساس نرسیده '0' برمی‌گرداند و تمامی زیروارپ‌ها منتظر آن می‌مانند. زمانی که آخرین زیروارپ به سدّ حساس برسد و این عبارت '1' برگرداند، تمامی زیروارپ‌ها در زمان‌بند به وضعیت آماده-ترکیب علامت زده می‌شوند.

برای اجرا دستور bar.synch_partner، تاخیر اجرای ۲۴ سیکل، برابر با عمق خط‌لوله، در نظر گرفته شده است.

۵-۵ نمونه عملکرد

در این بخش عملکرد DWR را با ارائه مثالی واضح‌تر می‌سازیم. برای این مثال از یکی از دو کرنل BFS استفاده می‌کنیم که در الگوریتم ۵-۴ نشان داده شده است. بخشی از این کد که علامت گذاری شده، با در الگوریتم ۵-۵ با نحو PTX نشان داده شده است. برای این مثال فرض شده است که سه زیروارپ (هرکدام با اندازه‌ای برابر پهنای SIMD) همزمان در حال اجرا هستند. مسیری که نخ‌های این زیروارپ‌ها اجرا می‌کنند در الگوریتم ۵-۵ نشان داده شده است. این زیروارپ‌ها با A، B، و C نشان داده شده‌اند و واگرایی انشعاب درون هریک از این زیروارپ‌ها رخ نداده است.

الگوریتم ۵-۴: کد کودا متعلق به کرنل محک BFS. قسمت‌های خط‌چین شده در مثال این بخش اجرا خواهند شد.

```
__global__ void
Kernel( Node* g_graph_nodes,
    int* g_graph_edges, bool* g_graph_mask,
    bool* g_updating_graph_mask, bool *g_graph_visited,
    int* g_cost, int no_of_nodes)
{
    int tid = blockIdx.x*MAX_THREADS_PER_BLOCK + threadIdx.x;
    if( tid<no_of_nodes && g_graph_mask[tid])
    {
        g_graph_mask[tid]=false;
        for(int i=g_graph_nodes[tid].starting;
    i<(g_graph_nodes[tid].no_of_edges + g_graph_nodes[tid].starting);
    i++)
        {
            int id = g_graph_edges[i];
            if(!g_graph_visited[id])
            {
                g_cost[id]=g_cost[tid]+1;
                g_updating_graph_mask[id]=true;
            }
        }
    }
}
```

در شکل ۵-۲ نحوه‌ی اجرای کد الگوریتم ۵-۵ توسط DWR و ماشین‌هایی با اندازه وارپ ثابت ارائه شده است تا مزیت استفاده از DWR را واضح سازد. در این مثال، برای دستورالعمل‌های دسترسی به حافظه سراسری (ld.global) تاخیر ۱۰ سیکل و برای باقی دستورالعمل‌ها تاخیر ۱ سیکل درنظر گرفته شده است و زمان‌بندی وارپ‌ها با سیاست راندروبین[۷] است.

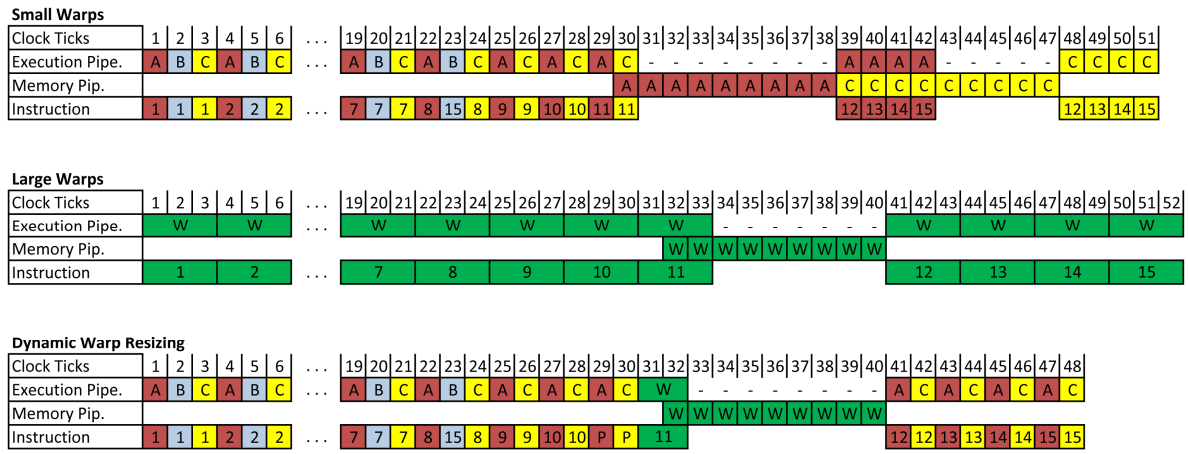
ماشین وارپ کوچک، هریک از این زیروارپ را مستقلا زمان‌بندی می‌کند. این ماشین مادامی که دسترسی به حافظه سراسری وجود ندارد، با درمیان کردن اجرای زیروارپ‌ها سیکل‌های بیکاری را کمتر می‌کند. بدین ترتیب در سیکل ۲۳ ام، اجرای وارپ B به پایان می‌رسد. اما در سیکل ۲۹ ام و هنگامی که زیروارپ A دستور دسترسی به حافظه را اجرا می‌کند، به مدت ۹ سیکل از وضعیت آماده خارج می‌شود. در سیکل ۳۰ ام زیروارپ C دستور حافظه را اجرا می‌کند و آن نیز به مدت ۹ + ۹ سیکل از وضعیت آماده خارج می‌شود. در بازه‌های زمانی سیکل ۳۱ تا ۳۸ و ۴۳ تا ۴۷ هسته هیچ زیروارپ آماده‌ای ندارد و بیکار می‌ماند و این به علت تاخیر اجرای درخواست‌های حافظه است. اگر زیروارپ‌های A و C در قالب یک وارپ بزرگ دستور حافظه را اجرا می‌کردند، احتمال الحاق دسترسی‌ها وجود داشت که خود می‌توانست تاخیر دسترسی به حافظه را کم کند.

[7] Round-robin

الگوریتم ۵-۵: کد PTX متناظر با بخش‌های علامت‌گذاری شده در الگوریتم ۴-۴. در سمت راست مسیرهای واگرایی و دستورالعمل‌هایی که هریک از وارپ A، B، و C اجرا می‌کنند نشان داده شده است.

```
                                                                A     B     C
    .entry Kernel (
            .param .u64 g_graph_nodes,
            .param .u64 g_graph_edges,
            .param .u64 g_graph_mask,
            .param .u64 g_updating_graph_mask,
            .param .u64 g_graph_visited,
            .param .u64 g_cost,
            .param .s32 no_of_nodes)
    {
    $LBB1__Kernel:
 1:     mov.u16        %rh1, %ctaid.x;                             *     *     *
 2:     mul.wide.u16   %r1, %rh1, 512;                             *     *     *
 3:     cvt.u32.u16    %r2, %tid.x;                                *     *     *
 4:     add.u32        %r3, %r2, %r1;                              *     *     *
 5:     ld.param.s32   %r4, [no_of_nodes];                         *     *     *
 6:     setp.le.s32    %p1, %r4, %r3;                              *     *     *
 7:     @%p1 bra       $Lt_0_5122;                                 *     *     *
 8:     cvt.u64.s32    %rd1, %r3;                                  *     -     *
 9:     ld.param.u64   %rd2, [g_graph_mask];                       *     -     *
10:     add.u64        %rd3, %rd2, %rd1;                           *     -     *
11:     ld.global.s8   %r5, [%rd3+0];                              *     -     *
12:     mov.u32        %r6, 0;                                     *     -     *
13:     setp.eq.s32    %p2, %r5, %r6;                              *     -     *
14:     @%p2 bra       $Lt_0_5122;                                 *     -     *
                                .
                                .
                                .
        $Lt_0_5122:
15:         exit;                                                  *     *     *
    }
```

ماشین وارپ بزرگ، سه زیروارپ را تجمیع می‌کند و همواره یک وارپ بزرگ را اجرا می‌کند. این ماشین واگرایی انشعاب در هر سه زیروارپ A، B، و C را در قالب یک پشته همگرایی مدیریت می‌کند. اگرچه هریک از این سه زیروارپ عاری از واگرایی انشعاب هستند، ولی قرار دادن آنها در یک وارپ بزرگ، واگرایی انشعاب ایجاد می‌کند زیرا نخ‌های زیروارپ B مسیر واگرایی متفاوتی نسبت به نخ‌های زیروارپ A و C دارند. تا دستور شماره ۷، ماشین‌های وارپ بزرگ و وارپ کوچک مشابه هم عمل می‌کنند. اما در دستور ۷ واگرایی انشعاب رخ می‌دهد. این واگرایی یک‌سویه است و طبق ساختار پشته ابتدا باید مسیر واگرایی (وارپ‌های A و C) اجرا شود و سپس نقطه بازهمگرایی (وارپ‌های A، B، و C). همانطوری که در شکل ۵-۲ نشان داده شده است، وارپ W با نخ‌های فعالی از زیروارپ‌های A و C اجرای دستورالعمل‌های ۸، ۹، و ۱۰ را ادامه می‌دهد تا جایی که به دستور دسترسی به حافظه ۱۱ می‌رسد. با الحاق دسترسی به حافظه‌ی زیروارپ‌های A و C، وارپ W به‌مدت ۹ سیکل از وضعیت آماده خارج می‌شود و منتظر اتمام واکشی داده از حافظه سراسری می‌شود. سپس اجرای زیروارپ‌های فعال A و C در دستورالعمل‌های ۱۲، ۱۳، و ۱۴ را ادامه می‌دهد. این ماشین نسبت به ماشین وارپ کوچک مدت زمان



شکل ۵-۲: اجرای کد الگوریتم ۵-۵ روی DWR، و ماشین‌هایی با اندازه وارپ ثابت بزرگ و کوچک. سه وارپ کوچک این مثال A، B، و C نام دارند و هنگامی که در قالب یک وارپ بزرگ اجرا شوند W نام می‌گیرند. برای رعایت اختصار، سیکل‌های ۷ تا ۱۸ نمایش داده نشده‌اند.

کمتری برای دسترسی به حافظه صرف می‌کند زیرا دسترسی به حافظه زیروارپ‌ها را الحاق کرده است. ولی در هنگام اجرا دستورالعمل‌های ۸، ۹، ۱۰، ۱۲، ۱۳، و ۱۴ دو زیروارپ فعال دارد درحالی که هر دستور را در سه سیکل اجرا می‌کند و یک سیکل در هر دستور هدر می‌شود. علت این سیکل اضافی، استفاده از وارپ بزرگ و رخ دادن واگرایی انشعاب است.

ماشین DWR، سه زیروارپ را مستقل از هم زمان‌بندی می‌کند ولی برای اجرای دستورالعمل‌های دسترسی به حافظه سراسری، ابتدا وارپ‌ها را قبل از دستور حافظه همگام می‌کند و سپس دستور حافظه را با یک وارپ بزرگ اجرا می‌کند. این عملیات سربار اجرای یک دستور اضافی برای همگام کردن زیروارپ‌ها را تحمیل می‌کند. همان‌طوری که در شکل ۵-۲ نشان داده شده است، DWR تا دستور ۱۰ مشابه ماشین وارپ کوچک عمل می‌کند. قبل از اجرای دستور ۱۱، دستور سدّ حساس برای همگام کردن وارپ‌ها اجرا می‌شود که این دستور در شکل با P نشان داده شده است (توجه این دستور در الگوریتم ۵-۵ نشان داده نشده است). هر زیروارپ پس اجرای این دستور از حالت آماده خارج می‌شود و منتظر دیگر زیروارپ‌های معتبر می‌ماند تا ۱) به سدّ حساسی برسند، یا ۲) به اتمام برسند. پس از اتمام انتظار روی سدّ حساس در سیکل ۳۰ ام، سیکل ۳۱ ام با اجرای وارپ بزرگ متشکل از زیروارپ‌های A و C ادامه پیدا می‌کند (توجه شود که زیروارپ B پیش‌تر به پایان رسیده است). این دستور دسترسی به حافظه‌های

زیروارپ‌های A و C را الحاق می‌کند و به مدت ۹ سیکل منتظر واکشی داده از حافظه سراسری می‌شود. سپس زیروارپ‌ها مستقلاً زمان‌بندی می‌شود تا اجرای همگی آنها به پایان برسد.

۵-۶ سربار سخت‌افزاری

زمان‌بند وارپ‌ها در معماری پایه قادر است وضعیت چندین وارپ را همزمان بروز کند (برای دستور syncthreads هنگام رها کردن از قفل لازم است). SCO چندین زیروارپ با وضعیت آماده-ترکیب را در قالب یک وارپ ایشو می‌کند. برای ساده کردن طراحی، SCO زیروارپ‌های آماده-ترکیب را در محدوده‌ی فاصله‌ی بین شناسه‌های معینی می‌یابد. این فاصله از طریق بیشینه اندازه وارپ، که پارامتر قابل پیکربندی DWR است، مشخص می‌شود. برای مثال، اگر بیشینه اندازه وارپ ۴ زیروارپ باشد، SCO زیروارپ‌هایی با شناسه‌های بین ix4 تا (i+1)x4-1 را وارسی می‌کند و i در محدوده تعداد وارپ‌های بزرگ تغییر می‌کند. محدوده زیروارپ‌های یک وارپ بزرگ، متناظراً با سدّ حساس در جدول PST همگام می‌شوند.

برای انجام عملیات صحیح، اندازه وارپ به همراه وارپ ایشو شده (و ماسک فعالیت نخ‌ها) باید عبور داده شود. اندازه وارپ (تعداد زیروارپ‌ها) می‌تواند مضربی از پهنای SIMD (اندازه زیروارپ) باشد. دانستن اندازه وارپ برای جلوی خطلوله لازم است تا واکشی و دیکود همزمان زیروارپ‌ها را انجام دهد. در پشت خطلوله، دانستن شناسه زیروارپ و ماسک فعالیت متناظرش کافی است تا عملوندهای ثباتی خوانده شوند، دستور اجرا شود، و در پایان ثبات‌ها پسنویسی شوند.

برای پشتیبانی از توسعه ISA در سخت‌افزار، یک سطر برای هر وارپ بزرگ در PST فرض می‌کنیم. با داشتن ۸ زیروارپ در هر وارپ بزرگ، هر سطر ۱ بیت اعتبار، ۳۲ بیت PC، و ۸ بیت بردار بیتی قفل خواهد داشت. برای ۱۶ وارپ بزرگ در هر هسته، اندازه PST، ۸۲ بایت در هر SM خواهد بود. یک مقایسه کننده ۳۲ بیتی برای مقایسه PC سطر با PC دستور سد نیاز است تا صورت برابر نبودن ILT را بروز کند. درحالی که ۱۱ عدد از محک‌های ارزیابی شده در این پژوهش هیچ سطری در ILT ذخیره نمی‌کنند، اندازه ILT تا ۳۶ سطر می‌تواند بزرگ شود (در محک MP). برای ILT جدول ۳۲ سطری که ۴ مجموعه‌ی ۸ راهی دارد را در نظر می‌گیریم که با دو بیت پایینی PC فهرست می‌شود. هر سطر از ILT ۱ بیتِ اعتبار و ۳۰ بیتِ تگ دارد. در نتیجه، اندازه ILT برابر ۱۲۴ بایت خواهد بود.

زمان‌بند وارپ‌ها، ۳۲ بیت PC، ۸ بیت ماسک فعالیت، و ۲ بیت وضعیت ذخیره می‌کند [۳۶]. هر سطر از زمان‌بند وارپ‌ها کمی بزرگ‌تر می‌شود تا ۳ بیت وضعیت را ذخیره کند تا ترکیب–آماده هم پشتیبانی شود. بافرض ۶۴ کیلوبایت رجیستر فایل، ۱۶ کیلوبایت حافظه مشترک، و ۴۸ کیلوبایت حافظه نهان داده در هر هسته، نیاز به فضای ذخیره‌ی PST و ILT کمتر از ۱٪ سربار به هر هسته تحمیل می‌کند.

فصل ۶

نتایج شبیه‌سازی

در این بخش نتایج بدست آمده از شبیه‌سازی از ارزیابی تکنیک‌های پیشنهادی در این پژوهش را ارائه می‌کنیم. در ابتدا محیط شبیه‌سازی را معرفی می‌کنیم. سپس معیارهای ارزیابی خود را بیان می‌کنیم. سپس نتایج ارزیابی CROWN را ارائه می‌کنیم. در بخش پایانی این فصل نتایج ارزیابی DWR ارائه می‌شود.

۶-۱ تنظیمات و محیط شبیه‌سازی

برای شبیه‌سازی معماری شبه GPU ی این پژوهش، از شبیه‌ساز GPGPU-sim نسخه‌ی 2.1.1b استفاده شده است. GPGPU-sim امکان شبیه‌سازی بارکاری کودا (و همچنین OpenCL) را روی یک پردازنده‌ی شبه GPU فراهم می‌کند و ما در این پژوهش مجموعه‌ای از کاربردهای کودا را برای ارزیابی DWR استفاده کردیم. برنامه‌های نوشته شده با مدل کودا به کتابخانه‌ی libcudart.so لینک می‌شوند و از طریق توابع موجود در این کتابخانه (که همان CUDA API هستند) GPU را کنترل می‌کنند. فرآیند نصب GPGPU-sim، کتابخانه‌ای جانشین برای libcudart.so ایجاد می‌کند و هنگامی که برنامه به این کتابخانه جدید لینک شوند، GPUی تحت‌کنترل آنها GPUی مجازی GPGPU-sim خواهد بود. در این حالت، کلیه فراخوانی‌های توابع libcudart، شبیه‌سازی می‌شوند. در GPGPU-sim، اجرای بارکاری روی هسته‌ها، تراکنش آنها با کنترل‌کننده‌های حافظه، و حافظه‌ی DRAM خارج از تراشه، سیکل به سیکل مدل می‌شود. پیکربندی‌های متفاوتی برای GPGPU-sim می‌توان در نظر گرفت تا محدوده وسیعی از معماری‌ها را مدل کند. در ادامه در رابطه با ساختار این شبیه‌ساز توضیحات بیشتری می‌دهیم.

جدول ۶-۱: پارامترهای پیکربندی GPGPU-sim در حالت پایه.

شبکه میان ارتباطی	
تعداد هسته‌ها	16
تعداد کنترل‌کننده‌های حافظه	6
تعداد SMهایی که یک واسط شبکه را به اشتراک می‌گذارند	2
اندازه بافر هر واسط شبکه	8
هسته	
اندازه وارپ	32 Threads
تعداد نخ‌ها در هر هسته	1024
بیشترین تعداد بلوک همزمان	8
تعداد رجیسترها	16K 32-bit
پهنای SIMD	8
اندازه حافظه مشترک	16KB
اندازه حافظه نهان داده	48KB : 12-way : LRU : 64BytePerBlock
اندازه حافظه نهان بافت	16KB : 2-way : LRU : 64BytePerBlock
اندازه حافظه نهان ثابت	16KB : 2-way : LRU : 64BytePerBlock
کلاک دهی	
هسته	1300 MHz
شبکه	650 MHz
حافظه DRAM	800 MHz
حافظه DRAM و کنترل کننده آن	
تعداد	8
سیاست زمان‌بندی DRAM	First-Come First-Serve
مشخصات رمان حافظه GDDR3	tRRD=12, tRCD=21, tRAS=13, tRP=34, tRC=9, tCL=10

GPGPU-sim کلیه واحدهای لازم برای مدل کردن بارهای کاری غیرگرافیکی، از جمله هسته‌های پردازشی، کنترل‌کننده‌های حافظه، شبکه‌ی میان ارتباطی روی تراشه، و حافظه‌ی DRAM خارج تراشه را داراست و واحدهای گرافیکی مثل رستراایز[1] که در GPUها وجود دارند ولی در کاربردهای همه‌منظوره‌ی نوشته شده با کودا استفاده نمی‌شوند را مدل نمی‌کند. شبیه‌ساز از ۲ بخش مجزا تشکیل شده است که

[1] Rasterizer

عبارتند از: ۱) جایگزین‌های CUDA API، و ۲) مدل معماری GPU. جایگزین‌های CUDA API وظیفه دارند بین کاربرد و معماری GPUی مدل شده واسط شوند و کلیه فراخوانی توابع موجود در libcudart را به‌صورت عملکردی[۲] مدل کنند. در این پژوهش، بخش مدل معماری GPU برای پیاده‌سازی معماری‌های پیشنهادی ویرایش شده است.

مدل معماری GPU در GPGPU-sim نسخه 2.1.1b بسیار نزدیک به معماری Tesla [۳۴] است و در سطح بالا از ۴ بخش تشکیل شده: ۱) هسته‌های پردازشی، ۲) شبکه میان ارتباطی بر تراشه، ۳) کنترل‌کننده‌های حافظه و حافظه‌ی DRAM خارج تراشه متناظر. هر بخش با نرخ کلاک قابل پیکربندی متفاوتی کار می‌کند و بخش‌ها از طریق بافر باهم در ارتباط هستند. پارامترهایی از قبیل فرکانس کاری، پهنای SIMD، اندازه حافظه مشترک، اندازه رجیسترفایل، اندازه حافظه نهان داده/ثابت/بافت، و عمق خط‌لوله، مهمترین پارامترهای قابل پیکربندی هر هسته هستند. تعداد کنترل‌کننده‌های حافظه، فرکانس کاری، تعداد بانک‌ها، پهنای بیتی خطوط داده، اندازه بافر زمان‌بندی، و سیاست زمان‌بندی درخواست‌ها، از مهمترین پارامترهای قابل پیکربندی برای کنترل‌کننده‌های حافظه هستند. زمان DRAM، و فرکانس کاری، از مهمترین پارامترهای قابل پیکربندی حافظه DRAM خارج تراشه هستند. بخش عمده‌ای از مدل شبکه میان ارتباطی از Booksim وارد شده است و با پارامترهای خاص این شبیه‌ساز پیکربندی می‌شود. در این پژوهش، الحاق برای دسترسی به حافظه داده، مشترک، و محلی مدل شده است. برای ارزیابی CROWN، الحاق دسترسی‌ها به حافظه در معماری پایه روی بخشی از وارپ (برابر با پهنای SIMD) مدل شده است و برای DWR، الحاق دسترسی‌ها به حافظه روی کل وارپ انجام می‌شود. برای مدل کردن پردازنده‌ای شبیه به Tesla، شبیه‌ساز را با پارامترهای داده شده در جدول ۶-۱ پیکربندی کرده‌ایم.

محک‌های سنجش معماری‌های شبه GPU، بارکاری‌های OpenCL و کودا هستند که محک‌هایی از مجموعه نمونه کدهای CUDA SDK 2.3 [۴۱]، مجموعه محک‌های استاندارد RODINIA [۵] و Parboil [۴۵] از متداول‌ترین آنها هستند. در این پژوهش ما محک‌هایی از این مجموعه‌ها جمع‌آوری کردیم. معیار انتخاب ما ۱) قابل اجرا بودن بارکاری در محیط شبیه‌سازی، و ۲) نمایش رفتارهای متفاوت بوده است. همچنین، محک‌هایی از مجموعه محک‌های منتشر شده به همراه GPGPU-sim و محک MUMmer-

[2] Functional

جدول ۶-۲: مشخصات محک‌های ارزیابی شده در این پژوهش.

نام	مخفف	ابعاد توری	ابعاد بلوک	تعداد کل دستورالعمل‌ها	تعداد بلوک‌نخ‌های همروند هر هسته
BFS Graph[5]	BFS	16x(8)	16x(512)	1.4M	1
Back Propagation[5]	BKP	2x(1,64)	2x(16,16)	2.9M	4
Coulumb Poten. [45]	CP	(8,32)	(16,8)	113M	8
Dyn_Proc[5]	DYN	13x(35)	13x(256)	64M	2
Gaussian Elimin.[5]	GAS	48x(3,3)	48x(16,16)	9M	1
Hotspot[5]	HSPT	(43,43)	(16,16)	76M	2
Fast Wal. Trans.[41]	FWAL	6x(32) 3x(16) (128)	7x(256) 3x(512)	11M	2
MUMmer-GPU++[17] big	MP2	(196)	(256)	139M	2
MUMmer-GPU++[17] small	MP	(1)	(256)	0.3M	1
Matrix Multiply[41]	MTM	(5,8)	(16,16)	2.4M	4
MUMmer-GPU[2] big	MU2	(196)	(256)	75M	4
MUMmer-GPU[2] small	MU	(1)	(100)	0.2M	1
Nearest Neighbor[5]	NNC	4x(938)	4x(16)	5.9M	8
N-Queen [2]	NQU	(256)	(96)	1.2M	1
Scan[41]	SC	(64)	(256)	3.6M	4
Needleman-Wun. [5]	NW	2x(1) … 2x(31) (32)	63x(16)	12M	2

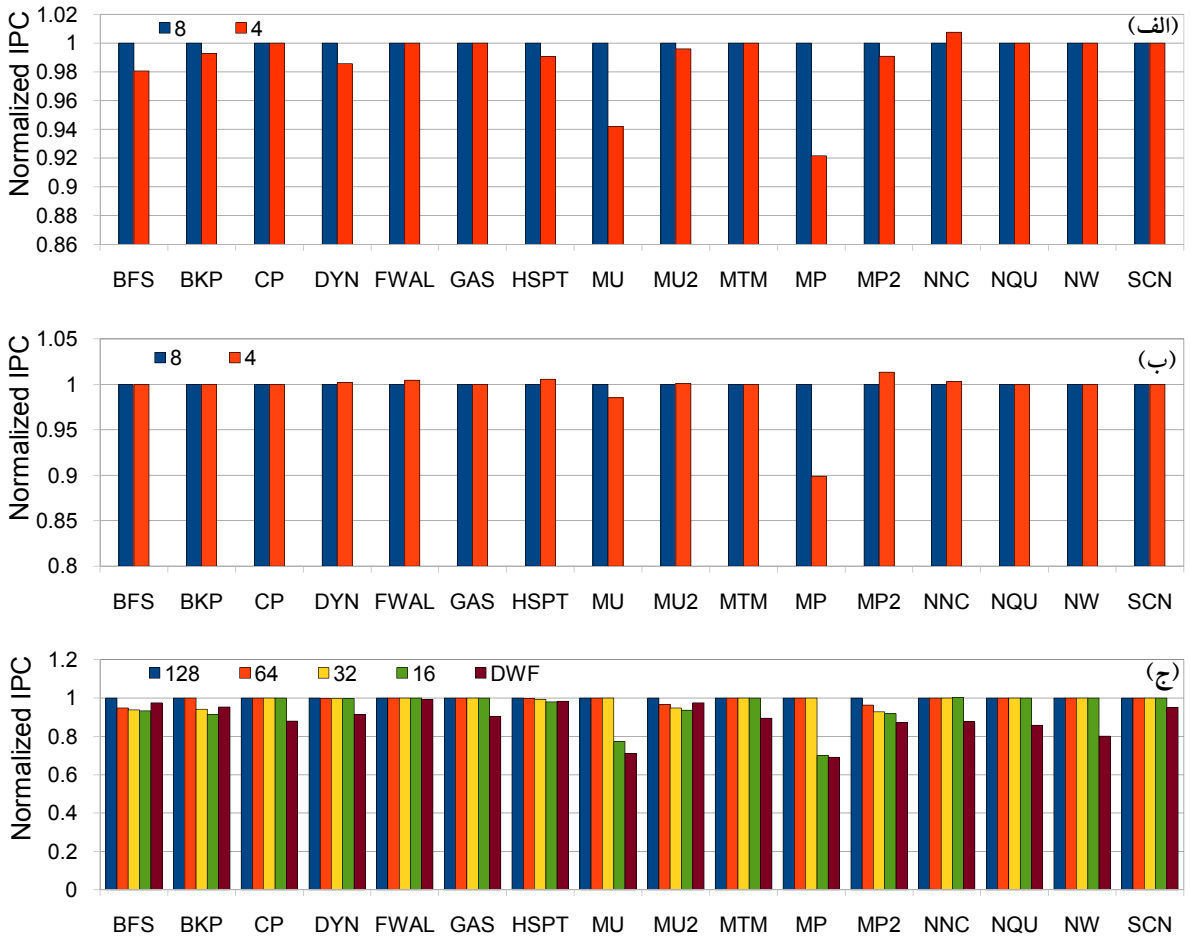
++GPU [۱۷] به عنوان یک برنامه کاربردی مطرح، برای ارزیابی معماری‌های پیشنهادی در این پژوهش استفاده شده‌اند. برخی از خصوصیات محک‌های این پژوهش در جدول ۶-۲ آورده شده‌اند.

۶-۲ معیار ارزیابی

معیار ارزیابی کارآمدی تکنیک‌های پیشنهادی در این پژوهش، گذردهی یا متوسط تعداد دستورالعمل‌های اجرا شده در هر سیکل است که Instruction Per Clock یا IPC نیز گفته می‌شود. برای آنکه مقایسه با معنی‌تر انجام شود از گذردهی نرمال شده یا Normalized IPC استفاده می‌کنیم که معیاری مناسب برای مقایسه گذردهی روش‌ها متفاوت در مقایسه با معماری پایه است.

۶-۳ نتایج CROWN

در این بخش نتایج شبیه‌سازی‌ها حاصل از ارزیابی عملکرد روش پیشنهادی CROWN را ارائه می‌کنیم. در ابتدا فضای طراحی CROWN را بررسی می‌کنیم و تاثیر پارامترهای پیکربندی CROWN بر کارایی آن را



شکل ۶-۱: نتایج ارزیابی گذردهی CROWN برای پیکربندی‌های مختلف آن. الف) حساسیت به تعداد سطرهای جدول گروه‌بندی پویای آماده، ب) حساسیت به تعداد سطرهای جدول گروه‌بندی پویای منتظر، و ج) حساسیت به تعداد سطرهای جدول سدهای بازهمگرایی.

مشاهده می‌کنیم. سپس روش پیشنهادی را با روش‌های گذشته مقایسه می‌کنیم. در پایان حساسیت یافته‌های خود به تغییر برخی از پارامترهای معماری پایه را ارزیابی می‌کنیم.

۶-۳-۱ فضای طراحی CROWN

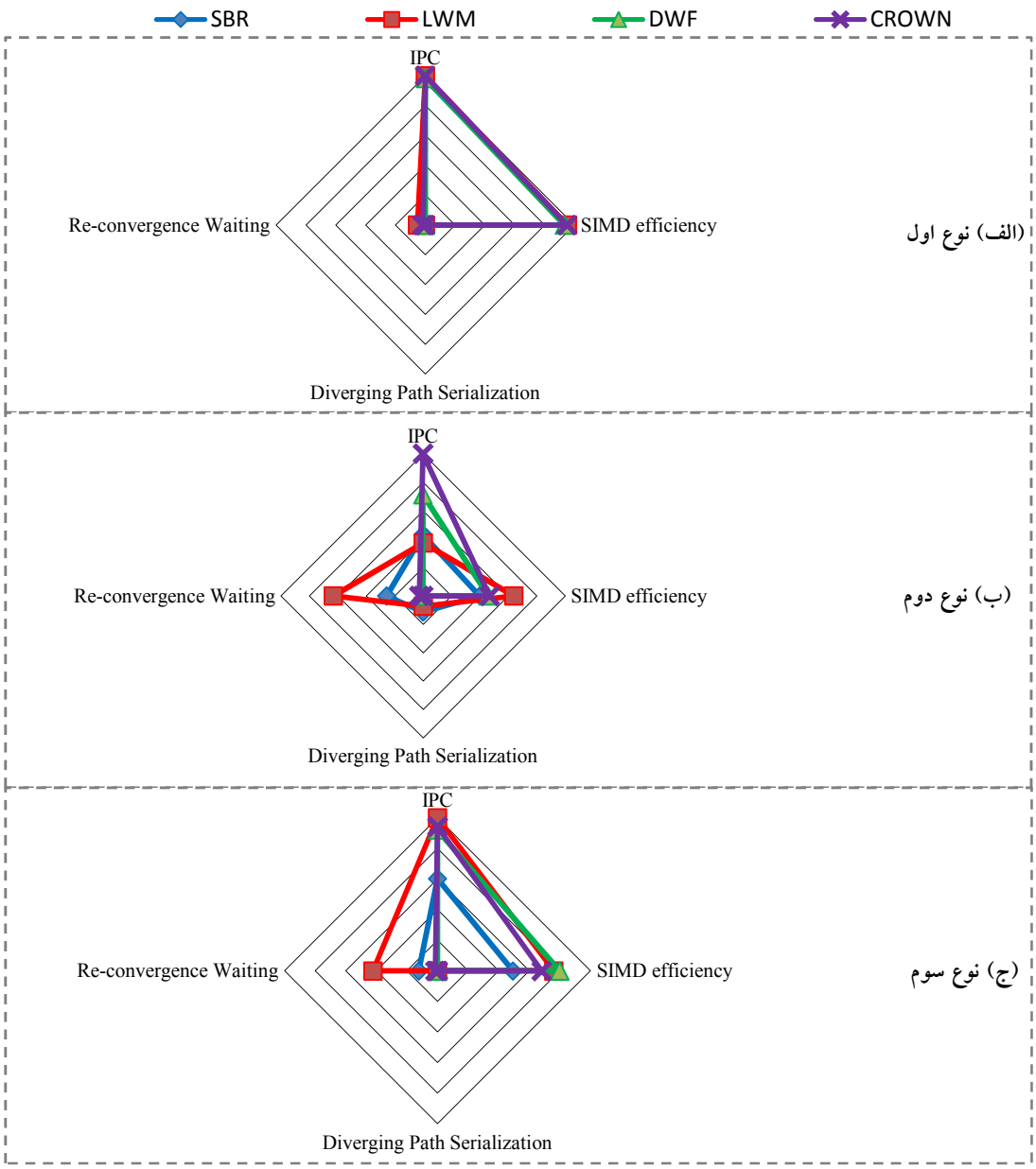
معماری پیشنهادی برای CROWN سه پارامتر قابل پیکربندی دارد که بر کارایی و هزینه سخت‌افزاری آن بسیار موثر هستند. این پارامترها عبارتند از تعداد سطرهای موجود در جداول ۱) گروه‌بندی پویای آماده، ۲) گروه‌بندی پویای منتظر، و ۳) سدهای بازهمگرایی. در بخش ۴-۳-۱ هزینه سخت‌افزاری پیکربندیی که ۸ سطر در جدول گروه‌بندی پویای آماده، ۸ سطر در جدول گروه‌بندی پویای منتظر، و ۱۲۸ سطر در جدول سدهای بازهمگرایی داشته باشد را ارزیابی کردیم. این پیکربندی را به عنوان پیکربندی پایه در ارائه نتایج

CROWN در نظر می‌گیریم و در این بخش تاثیر کاهش تعداد سطرهای هریک از این جداول را بررسی می‌کنیم. شکل ۶-۱ نتایج این ارزیابی را ارائه می‌کند.

شکل ۶-۱ (الف) حساسیت به اندازه جدول گروه‌بندی پویای آماده را گزارش می‌کند. کاهش سطرهای این جدول امکان ترکیب وارپ‌های واگراشده و به تبع کاهش خطوط خالی وارپ‌های واگراشده را کم‌تر می‌کند. این امر باعث می‌شود نرخ فعالیت SIMD افت کند که کاهش گذردهی را به‌دنبال دارد. همانطوری که در این شکل دیده می‌شود، کاهش اندازه جدول از ۸ به ۴ می‌تواند گذردهی را تا ۸٪ کاهش دهد. توجه شود که حساسیت به کاهش اندازه این جدول تنها در بارکاری‌هایی که واگرایی انشعاب در آنها رخ می‌دهد می‌تواند دیده می‌شود و در اغلب این موارد (به غیر از MU و MP) کارایی کمتر از ۳٪ کاهش می‌یابد.

در شکل ۶-۱ (ب) حساسیت به کاهش اندازه جدول گروه‌بندی پویای منتظر ارزیابی شده است. به طریقی مشابه با جدول گروه‌بندی پویای آماده، کاهش اندازه این جدول نیز احتمال ترکیب و به تبع نرخ فعالیت SIMD را پایین می‌آورد. همان‌طوری که گزارش شده، کاهش اندازه این جدول از ۸ به ۴ در اغلب موارد گذردهی را به میزان ناچیزی تغییر می‌دهد. در بین بارکاری‌های ارزیابی شده، گذردهی MP بیش از باقی به اندازه این جدول وابسته است.

شکل ۶-۱ (ج) حساسیت به اندازه جدول سدهای بازهمگرایی را ارزیابی می‌کند. کاهش تعداد سدهای بازهمگرایی پشتیبانی‌شده، میزان انتظار روی نقطه بازهمگرایی را کم می‌کند. اما از طرف دیگر نرخ فعالیت SIMD را کم می‌کند زیرا ممکن است نخ‌های واگراشده هرگز همگرا نشوند و سراسر کرنل با وارپ‌های نیمه‌پر اجرا شود. تحلیل‌های ما نشان می‌دهد که تعداد سدهای مورد نیاز تا ۱۸۸ (در MU2)، ۱۳۸ (در MP2)، و ۸۵ (در BFS) می‌تواند رشد کند و در باقی محک‌ها کمتر از ۶۴ سد نیاز است. همانطوری که در شکل ۶-۱ (ج) نشان داده شده است، کاهش سدهای بازهمگرایی همواره گذردهی را کم کرده است. کاهش سدهای بازهمگرایی باعث افزایش وارپ‌های نیمه‌پر می‌شود که رفته رفته سطرهای جدول سطح دوم را پر می‌کنند. به عنوان مثال در MU2، کاهش تعداد سدهای بازهمگرایی از ۱۲۸ به ۶۴، ۳۲، و ۱۶ تعداد وارپ‌های موردنیاز در سطح دوم را از ۴۳۲ به ۷۱۴، ۸۰۲، و ۸۹۰ می‌رساند. بدین ترتیب، افزایش سطرهای جدول سدهای همگرایی باعث کاهش سطرهای موردنیاز در جدول سطح دوم و همچنین افزایش گذردهی هسته می‌شود. همانطوری که در شکل ۶-۱ (ج) نشان داده شده، هنگامی که روش CROWN با



شکل ۶-۲: انواع رفتارها در محک‌های ارزیابی شده بر اساس ارتباط سه چالش واگرایی انشعاب با گذردهی. نوع اول: درجه کمی از رخداد واگرایی انشعاب. نوع دوم: تعدّد رخداد واگرایی انشعاب، درجه موازی سازی کم، و حساس به تعداد نخ‌های فعال. نوع سوم: تعدّد رخداد واگرایی انشعاب، درجه موازی‌سازی بالا، و حساس به نرخ فعالیت SIMD.

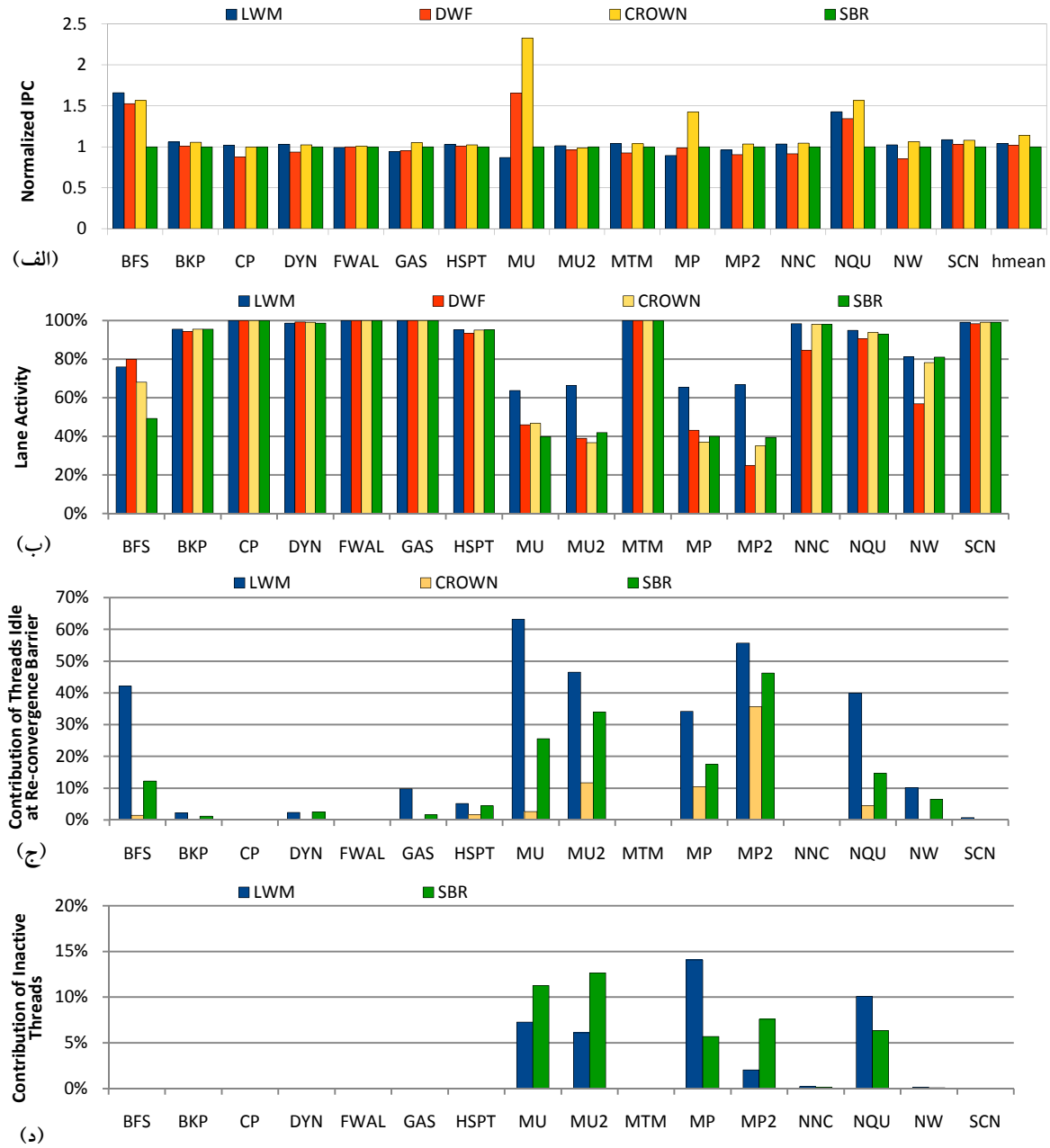
تعداد کمی سدّ بازهمگرایی پیکربندی شود، عملکردی نزدیک به روش DWF [۱۳] خواهد داشت و تفاوت آنها به ۱) اندازه وارپ، ۲) اندازه جدول‌های گروه‌بندی، و ۳) سیاست زمان‌بندی وارپ‌ها محدود می‌شود.

۶-۳-۲ مقایسه با کارهای قبلی

در این بخش کارایی CROWN را با روش‌های SBR، DWF [۱۳]، و LWM [۳۸] مقایسه می‌کنیم. برای DWF از سیاست Majority به عنوان Issue Heuristic استفاده شده (زیرا به‌طور متوسط بهترین عملکرد را دارد) و همچنین ظرفیت جدول LUT در آن برابر ۲۵۶ وارپ ۳۲ نخی (در ۲ مجموعه ۱۲۸ وارپی) پیکربندی شده است. برای LWM، اندازه وارپ بزرگ برابر ۱۲۸ نخ درنظر گرفته شده است.

محک‌های ارزیابی شده در این پژوهش را می‌توان به سه دسته تقسیم کرد. **نوع اول**، محک‌هایی هستند که رخداد واگرایی انشعاب در آنها کم/ناچیز است و یا کاربرد memory-bound هستند که در نتیجه روش‌هایی که برای مقابله با واگرایی انشعاب پیشنهاد شده‌اند (مثل DWF، LWM، و CROWN) مشابه روش پایه (SBR) عمل می‌کنند. **نوع دوم**، محک‌هایی هستند که رخداد واگرایی انشعاب در آنها چشمگیر است و تعداد نخ‌های همروند در هر هسته کم است. کمبود نخ می‌تواند به علت ۱) استفاده زیاد هر بلوک نخ از رجیسترفایل یا حافظه مشترک باشد یا ۲) بارکاری میزان کمی بلوک‌نخ بکار می‌گیرد. در این حالت وجود نخ‌های فعال اهمیت زیادی دارد و برای بهبود گذردهی، «کاهش نخ‌های غیرفعال» و «کاهش نخ‌های منتظر روی نقطه بازهمگرایی»، به میزان اهمیت «بهبود نرخ فعالیت SIMD» مهم خواهند شد. **نوع سوم**، محک‌هایی هستند که رخداد واگرایی انشعاب در آنها چمگیر است و تعداد نخ‌های همروند در هر هسته زیاد است. در این حالت «کاهش نخ‌های غیرفعال» و «کاهش نخ‌های منتظر روی نقطه بازهمگرایی» مرتبه کمتری از اهمیت نسبت به «بهبود نرخ فعالیت SIMD» دارند. علت آن است که در هسته نخ‌های همروند فراوانی وجود دارند و توقف «نخ‌های غیرفعال» و «نخ‌های منتظر روی نقطه بازهمگرایی» کمتر دیده می‌شود زیرا اغلب این توقف‌ها توسط دیگر نخ‌ها مخفی می‌شوند. این دسته‌بندی سه‌گانه در شکل ۶-۲ نشان داده شده است. محک‌هایی از جمله MU، MP، و NQU در نوع دوم، BFS در نوع سوم، و باقی محک‌ها در نوع اول قرار می‌گیرند.

شکل ۶-۳ پارامترهایی از قبیل گذردهی، نرخ فعالیت SIMD، انتظار روی نقطه بازهمگرایی، و نخ‌های غیرفعال را برای CROWN و روش‌های پیشین در هر محک گزارش می‌کند. در اغلب محک‌های نوع سوم، LWM، SBR و CROWN مشابه هم عمل می‌کنند و گذردهی DWF نسبت به آنها کمتر است. علت عملکرد ضعیف DWF در این مواقع، فقدان مکانیزمی برای همگرایی پس از اجرای مسیرهای واگرایی



شکل ۶-۳: مقایسه SBR، DWF، CROWN، و LWM برای محک‌های این پژوهش. الف) گذردهی (نرمال‌شده نسبت به SBR)، ب) نرخ فعالیت SIMD، ج) انتظار روی نقطه بازهمگرایی، د) نخ‌های غیرفعال.

است که باعث فراگیر شدن بهره‌برداری کم از SIMD حتی پس از نقطه بازهمگرایی می‌شود که گذردهی در DWF را کم می‌کند. برای کاربردهای نوع سوم انتظار می‌رود که LWM بهتر از باقی عمل کند زیرا با همگام نگهداشتن تعداد زیادی نخ و فشرده کردن بالای پشته همگرایی، نرخ فعالیت SIMD (که پارامتر حیاتی در این نوع کاربردها است) را بیشینه می‌کند. این مورد را می‌توان در محک BFS مشاهده کرد. برای محک‌های نوع دوم، بیشینه کردن نرخ فعالیت SIMD برای بیشینه کردن گذردهی کافی نیست و باید میزان

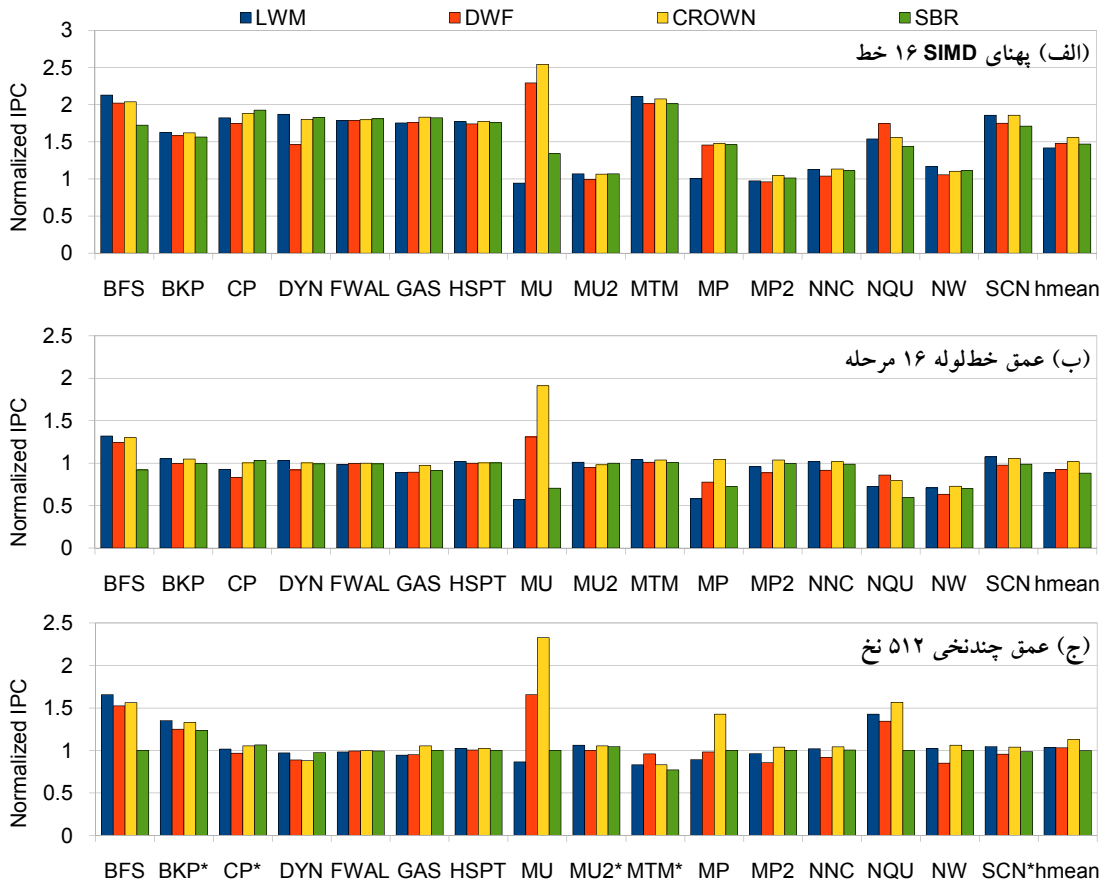
نخ‌های فعال را نیز افزایش داد. همانطوری که مشاهده می‌شود، CROWN برای محک‌های نوع دوم بهتر از باقی روش‌ها عمل می‌کند که این مهم بوسیله افزایش نخ‌های فعال از طریق ۱) کاهش «نخ‌ها منتظر روی نقطه همگرایی» و ۲) از بین بردن «نخ‌های غیرفعال» حاصل شده است. به طور متوسط، CROWN نسبت به SBR، DWF، و LWM گذردهی را ۱۴٪، ۱۲٪، و ۱۰٪ بهبود می‌بخشد.

۶-۳-۳ ارزیابی حساسیت

در این بخش حساسیت یافته‌های بخش ۶-۳-۲ به تغییر پارامترهایی از قبیل پهنای SIMD، عمق خطلوله، و عمق چندنخی (تعداد نخ‌های همروند در هر هسته) را می‌سنجیم. در معماری پایه پهنای SIMD برابر ۸ خط، عمق خطلوله برابر ۸ مرحله، و عمق چندنخی برابر ۱۰۲۴ نخ لحاظ شد. شکل ۶-۴ حساسیت گذردهی به پهنای SIMD وقتی از ۸ به ۱۶ خط تغییر پیدا کند، عمق خطلوله وقتی از ۸ به ۱۶ مرحله تغییر پیدا کند، و عمق چندنخی وقتی از ۱۰۲۴ به ۵۱۲ نخ تغییر پیدا کند، را گزارش می‌کند.

افزایش پهنای SIMD، گذردهی محک‌های compute-bound را افزایش می‌دهد و تغییر ناچیزی بر محک‌های memory-bound (مانند MU2 و MP2) دارد. کارایی روش CROWN با افزایش پهنای SIMD به دو دلیل محدودتر می‌شود: ۱) افزایش پهنای SIMD برای روش‌های LWM و CROWN که وارپ‌هایی به پهنای SIMD ایجاد می‌کنند، تداخل خط بیشتری هنگام گروه‌بندی نخ‌های واگرا شده ایجاد می‌کند. این امر باعث می‌شود که این روش‌ها (در مقایسه با SBR) در SIMD های پهن به «بهبود نرخ فعالیت SIMD» کمتری نسبت SIMD باریک‌تر نایل شوند. ۲) در CROWN سدهای بازهمگرایی در اندازه‌ای برابر با پهنای SIMD اعمال می‌شوند. افزایش پهنای SIMD باعث می‌شود که CROWN نخ‌های بیشتری را روی نقطه بازهمگرایی منتظر هم نگه‌دارد به تبع این امر باعث می‌شود تعداد نخ‌های فعال نسبت به SIMD باریک‌تر کمتر شود.

در رابطه با افزایش عمق خطلوله، دو مورد را بیان می‌کنیم. مورد اول، مشترک بین تمامی روش‌ها است که از ماهیت پردازنده‌های چندنخی ناشی می‌شود و مورد دوم خاصِ روش‌های گروه‌بندی پویا (مثل DWF و CROWN) است. **۱) مورد مشترک:** گذردهی در پردازنده‌هایی چندنخی بسیار عمیق (مانند GPU) زمانی به عمق خطلوله (تاخیر اجرای دستور محاسباتی) وابسته خواهند بود که تعداد نخ‌های همروند برای پرکردن خطلوله کافی نباشد. در این حالت افزایش عمق خطلوله کارایی را بسیار کم می‌کند. این مورد برای



شکل ۶-۴: حساسیت گذردهی به تغییر پارامترهای معماری پایه که با پهنای SIMD ۸ خط، عمق خطلوله ۸ مرحله، و عمق چندنخی ۱۰۲۴ نخ ارزیابی شد. الف) افزایش پهنای SIMD، ب) افزایش عمق خطلوله، و ج) کاهش عمق چند نخی. کلیه اعداد به ماشین SBR معماری پایه نرمال شده‌اند.

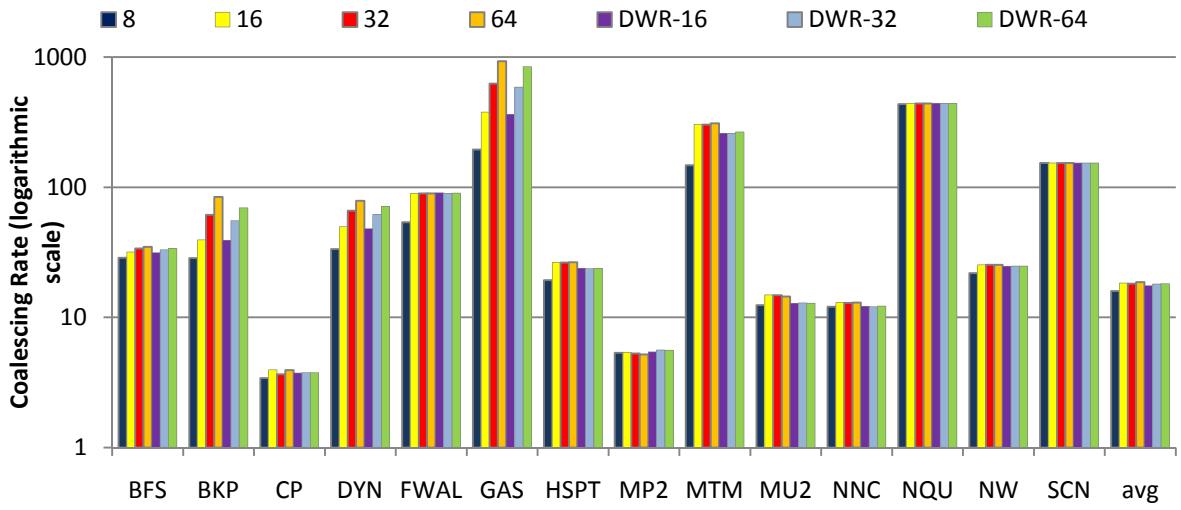
محک‌های MU، MP، NQU، و NW در شکل ۶-۴ (ب) دیده می‌شود که SBR نسبت به قبل گذردهی کمتری داشته است. در باقی موارد عمق خطلوله میزان ناچیزی بر گذردهی اثرگذار است. **۲) مورد خاص روش‌های گروه‌بندی پویا:** در روش‌های گروه‌بندی پویا (مثل DWF و CROWN) زمانی که وارپ آماده‌ای برای اجرا وجود نداشته باشد، وارپی نیمه‌پر از جدول گروه‌بندی خارج می‌شوند تا تاخیر باقی نخ‌ها را پر کنند. در این موقعیت، گروه‌بندی وارپ‌ها و ترکیب آنها محدودتر می‌شود زیرا وارپ‌های نیمه‌پر مدت کوتاه‌تری می‌توانند در جدول گروه‌بندی بمانند و در نتیجه به‌ندرت با دیگر وارپ‌ها ترکیب می‌شوند.

با کاهش عمق چندنخی، رفتار مشابهی بین روش‌های مختلف دیده می‌شود. شکل ۶-۴ (ج) حساسیت به کاهش عمق چندنخی از ۱۰۲۴ به ۵۱۲ نخ را گزارش می‌کند. توجه شود که محک‌های مثل BKP، CP، MU2، MTM، و SCN بیش از ۵۱۲ نخ همروند در هر هسته به کار می‌گیرند (با ستاره‌ای در کنار نام آنها

در شکل مشخص شده‌اند) و باقی محک‌ها به دلیل ۱) کمبود نخ همروند (BFS، DYN، FWAL، GAS، MU، MP، و NW)، ۲) استفاده زیاد حافظه مشترک در هر بلوک نخ (NQU)، ۳) استفاده زیاد از رجیسترفایل در هر بلوک نخ (HSPT، و MP2)، یا ۴) محدودیت هسته در بکار گرفتن کمتر از ۸ بلوک نخ همروند (NNC)، کمتر از ۵۱۲ نخ همروند در هر هسته دارند (و کاهش تعداد نخ‌های همروند کارایی آنها را تحت تاثیر قرار نمی‌دهد). انتظار می‌رود با کاهش عمق چندنخی، کارایی کاهش پیدا کند زیرا نخ‌های کمتری برای مخفی کردن تاخیر دسترسی به حافظه در هسته وجود خواهند داشت. در بین ۵ محک مذکور که به عمق چندنخی می‌توانند حساس باشند، با کاهش عمق چندنخی گذردهی SCN میزان ناچیزی کاهش یافته و در MTM این کاهش بسیار زیادتر است. بر خلاف انتظار اولیه، رفتار متفاوتی نیز دیده می‌شود که با کاهش عمق چندنخی، گذردهی CP و MU2 میزان ناچیزی افزایش یافته است و البته این افزایش در BKP مقدار چشمگیری است. علت آن است که حافظه خارج تراشه در BKP با درخواست‌های همزمان و متعددی از هسته‌های مختلف روبرو است که تاخیر دسترسی به حافظه را بالا برده است. هنگامی که عمق چندنخی پایین می‌آید، تعداد دسترسی‌های همزمان کم می‌شود و تاخیر دسترسی به حافظه کمتر می‌شود. این امر باعث بهبود گذردهی در این محک شده است.

۴-۶ نتایج **DWR**

در این بخش، کارایی روش DWR در مقایسه با پردازنده‌هایی که اندازه وارپ ثابتی دارند، می‌سنجیم. در DWR سه پارامتر قابل پیکربندی وجود دارد: ۱) اندازه ILT، ۲) اندازه کوچکترین وارپ، و ۳) اندازه بزرگترین وارپ. برای ILT حافظه‌ای از جنس حافظه نهان با ۳۲ سطر در ۴ مجموعه فرض می‌کنیم و اندازه کوچکترین وارپ در DWR را برابر پهنای SIMD در نظر می‌گیریم. بزرگترین اندازه وارپ را برای مقادیر ۱۶، ۳۲، و ۶۴ می‌سنجیم که به ترتیب با DWR-16، DWR-32، و DWR-64 نمایش داده می‌شوند. در بخش اول، نرخ الحاق دسترسی‌ها به حافظه را گزارش می‌کنیم. درصد سیکل‌های بیکاری در بخش دوم، و گذردهی در بخش سوم گزارش می‌شود. در بخش چهارم، حساسیت گذردهی نسبت به پارامترهایی از قبیل اندازه حافظه نهان داده، پهنای SIMD، و اندازه ILT گزارش می‌شود.

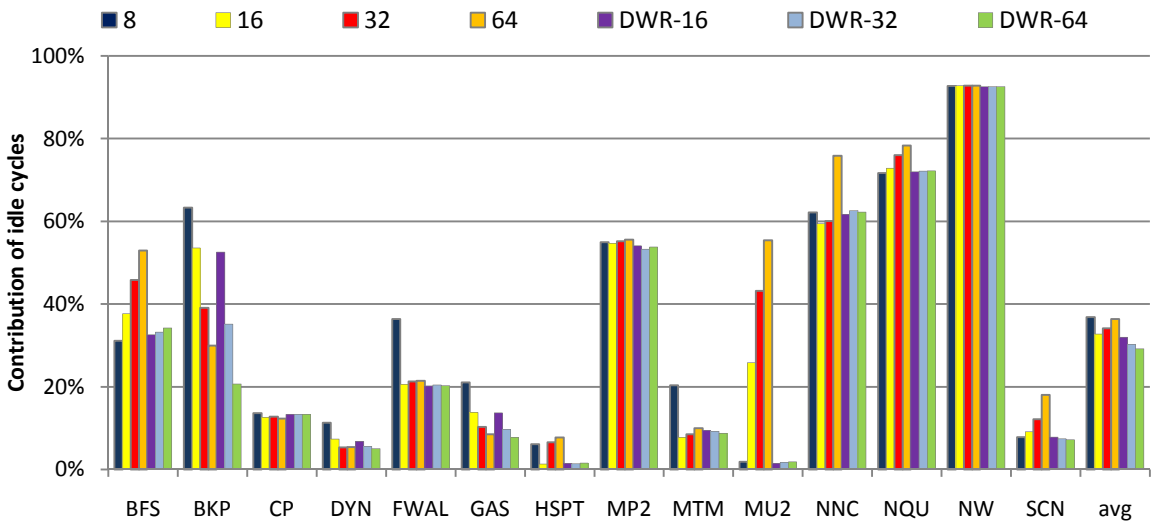


شکل ۶-۵: مقایسه نرخ الحاق بین پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.

۶-۴-۱ الحاق دسترسی‌ها به حافظه

شکل ۶-۵ نرخ الحاق دسترسی‌ها به حافظه را گزارش می‌کند. همان‌طوری که گزارش شده است، ماشین ۶۴ نخ ثابت در وارپ بیشترین نرخ الحاق را در اغلب محک‌ها دارد. DWR اکثر دستورالعمل‌ها را با وارپ‌های ۸ نخی (زیروارپ) اجرا می‌کند تا از همگام کردن‌های غیرالزامی جلوگیری کند. DWR برای آنکه به نرخ الحاق دسترسی‌ها به حافظه‌ای نزدیک به ماشین‌های اندازه وارپ بزرگ نایل شود، زیروارپ‌ها را در هنگام اجرای دسترسی به حافظه همگام می‌کند. در محک‌هایی که الحاق دسترسی به حافظه‌ها با تغییر اندازه وارپ تاثیرپذیر است (از جمله محک‌های BKP، DYN، GAS، MTM)، این مکانیزم موثر عمل می‌کند و نرخ الحاق را به میزان ۱۴٪ نسبت به ماشین ۸ نخی ثابت بهبود می‌دهد. مقایسه نرخ الحاق DWR با ماشین ۶۴ نخی ثابت، نشان می‌دهد که DWR به ۹۷٪ نرخ الحاق این ماشین دست یافته است.

با ماشین DWR، MU نرخ الحاق قابل ملاحظه‌ای را نسبت به ماشین‌ها «ثابت با وارپ بزرگ» از دست داده است. در این محک، مقدار قابل ملاحظه‌ای از دستورالعمل‌های حساس در ILT ذخیره می‌شوند. این درحالی است که این الحاق از دست رفته، منتهی به کاهش گذردهی نمی‌شود. علت آن است که ILT به میزان قابل توجهی از سربار همگام کردن‌های ناشی از واگرایی حافظه، می‌کاهد که متناظراً از سیکل‌های بیکاری کاسته شده است.

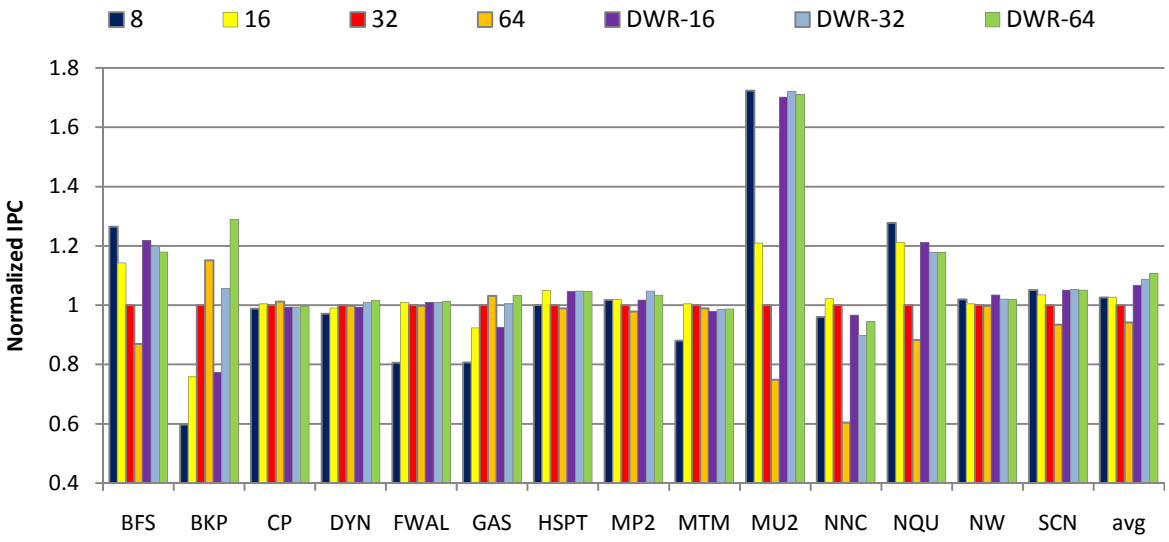


شکل ۶-۶: مقایسه درصد سیکل‌های بیکاری بین پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.

۶-۴-۲ سیکل‌های بیکاری

همان‌طوری که در فصل ۲ بحث شد، وارپ‌های کوچک سیکل‌های بیکاری را با کاهش انتظارهای ناشی از واگرایی انشعاب و واگرایی حافظه، کاهش می‌دهد. اما این کاهش سیکل‌های بیکاری ناشی از وارپ‌های کوچک، با از دست رفتن الحاق دسترسی به حافظه تاثیر می‌پذیرد و گاهی بی‌اثر می‌شود. DWR این نقص را با همگام کردن زیروارپ‌ها در هنگام اجرای دستور حافظه جبران می‌کند. DWR همگام کردن‌های غیرالزامیِ تمامی نخ‌های وارپ را کاهش می‌دهد و اجرای زیروارپ‌ها را همپوشانی می‌دهد تا تاخیر یکدیگر را به‌گونه موثری پنهان کنند. همان‌طوری که در شکل ۶-۶ نشان‌داده شده است، به طور متوسط استفاده از DWR-64 سیکل‌های بیکاری را ۲۶٪، ۱۲٪، ۱۷٪، و ۲۵٪ به ترتیب در مقایسه با ماشین‌هایی با اندازه وارپ ثابت ۸، ۱۶، ۳۲، و ۶۴ کاهش می‌دهد. همانطوری که در شکل ۶-۶ نشان داده شده است، DWR-64 به‌طور متوسط کمترین درصد سیکل بیکاری را دارد.

همگام کردن متوالی نخ‌های درون یک بلوک (از طریق سنکرون کننده‌های درون بلوکی) مانع از پیشرفت بیشتر زیروارپ‌ها و مخفی کردن تاخیر به میزان بیشتر می‌شود. برای مثال می‌توان به همگام کردن‌های غیرالزامی تمام نخ‌های بلوک در هر تکرار حلقه در MTM اشاره کرد. این همگام کردن‌ها مانع از آن می‌شود که DWR به گونه‌ی موثرتری سیکل‌های بیکاری بین تکرارهای حلقه را مخفی کند.



شکل ۶-۷: مقایسه گذردهی بین پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.

۶-۴-۳ گذردهی

شکل ۶-۷ گذردهی را برای DWR و پردازنده‌هایی با اندازه وارپ ثابت گزارش می‌کند. در اکثر محک‌ها، گذردهی DWR-64 نزدیک به ماشین اندازه وارپ ثابتی است که بهترین گذردهی را دارد. علت آن است که DWR مزایای کارایی هردو وارپ بزرگ و کوچک را مهیا می‌کند. به‌طور متوسط، DWR-64 کارایی را ۸٪، ۸٪، ۱۱٪، و ۱۸٪ به ترتیب در مقایسه با ماشین‌های ثابت ۸، ۱۶، ۳۲، و ۶۴ نخ در وارپ، بهتر می‌کند.

مهم است که مشاهده شود چرا در برخی از محک‌ها، ماشین‌های ثابت بهتر از DWR عمل می‌کنند. به عنوان مثال در NNC، ۱۷ دستور حساس در کل کرنل وجود دارد. اغلب این دستورالعمل‌ها در عمقی یکسان از واگرایی ولی در داخل مسیرهای واگرایی متفاوتی هستند. واگرایی و ترتیب زمان‌بندی وارپ‌ها باعث می‌شود تمامی این ۱۷ دستور در ILT درج شوند. بنابراین، DWR تعداد زیادی دسترسی‌های قابل الحاق شدن خارج از اندازه زیروارپ را از دست می‌دهد و به گذردهی نزدیک به ماشین ۸ نخی می‌رسد.

۶-۴-۴ ارزیابی حساسیت

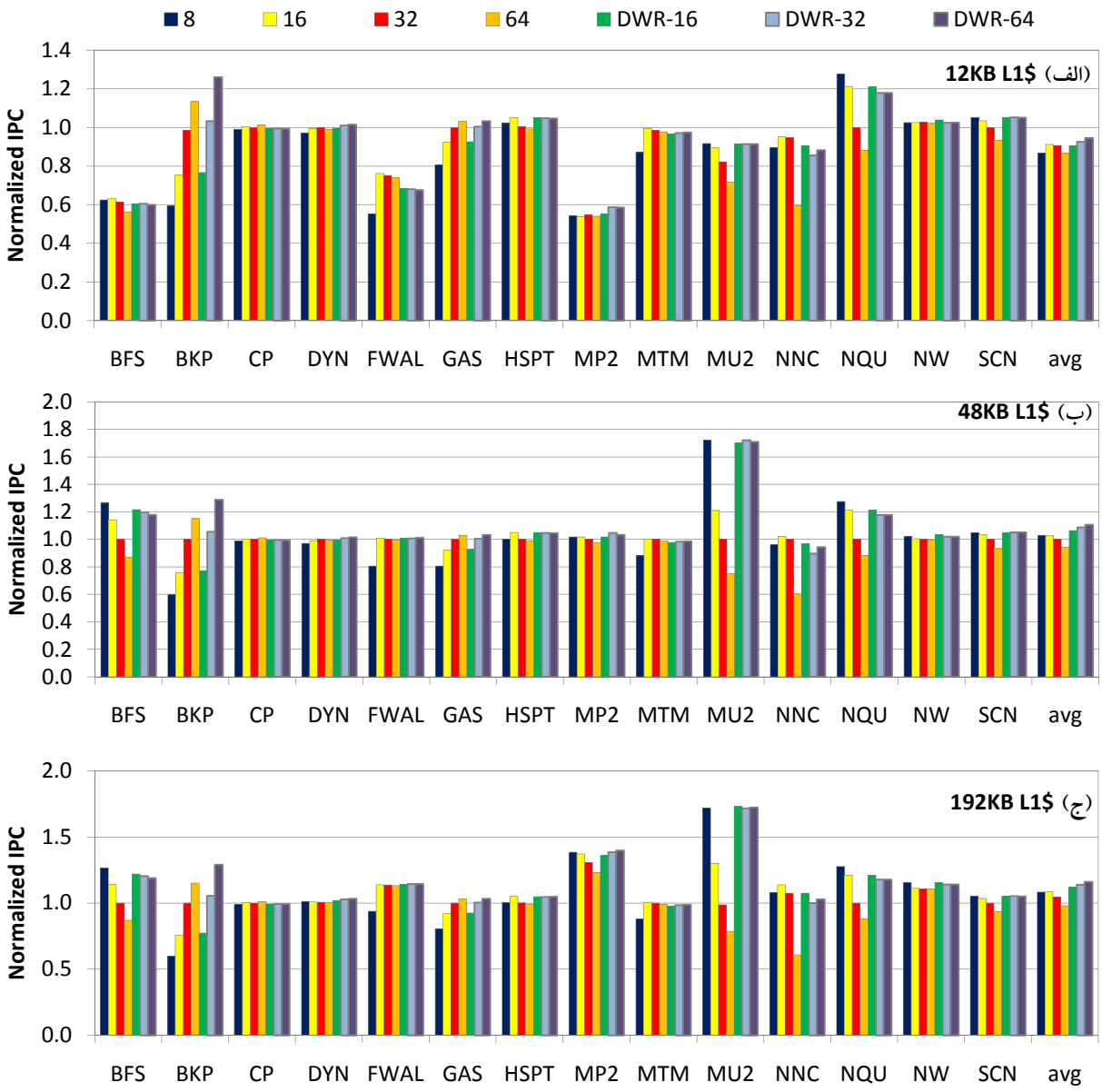
در این بخش، حساسیت گذردهی به پارامترهای معماری گسترده‌ای از جمله اندازه حافظه نهان داده، پهنای SIMD، و اندازه ILT را می‌سنجیم.

۶-۴-۴-۱ حافظه نهان داده

معماری پایه از ۴۸ کیلوبایت (۶۴ مجموعه، ۱۲ راه) حافظه نهان در هر هسته استفاده می‌کند. شکل ۶-۸ گذردهی DWR را در مقایسه با ماشین‌هایی با اندازه وارپ ثابت برای اندازه حافظه نهان‌های متفاوتی گزارش می‌کند؛ حافظه نهان‌های ۴ برابر کوچکتر (۱۲ کیلوبایت، ۳۲ مجموعه، ۶ راه) و ۴ برابر بزرگتر (۱۹۲ کیلوبایت، ۱۲۸ مجموعه، ۲۴ راه). همان‌طوری که گزارش شده است، استفاده از حافظه نهان کوچکتر، افزایش گذردهی ناشی از DWR را کاهش می‌دهد. این پدیده دو دلیل دارد. اول، واگرایی انشعاب اهمیتش را از دست می‌دهد زیرا در این شرایط به علت نرخ فقدان بالاتر، محک‌ها بیشتر memory-bound می‌شوند و کمتر compute-bound خواهند بود. این باعث می‌شود که اثر «کاهش واگرایی انشعاب» که توسط DWR انجام می‌شود روی گذردهی چشمگیر نباشد. این توضیح، گذردهی MU را توجیح می‌کند که در آن وارپ‌های کوچک نمی‌توانند گذردهی را برای حافظه نهان‌های کوچک بهتر کنند. دوم، حافظه نهان‌های کوچک، اثر «کاهش واگرایی حافظه» ناشی از DWR را کم می‌کنند زیرا اکثر دسترسی‌ها به حافظه نهان با فقدان روبرو می‌شوند که به نوبه‌ی خود واگرایی حافظه کمتر شده و بیشتر وارپ‌ها برای تمامی نخ‌هایشان به‌صورت یکپارچه با فقدان مواجه می‌شوند. از طرف دیگر، افزایش گذردهی با حافظه نهان‌های بزرگتر به طریقی مشابه توجیح می‌شود. تاثیر اندازه حافظه نهان بر محک‌هایی مثل NNC ناچیز است. فاصله‌ی گذردهی بین بهترین ماشین اندازه وارپ ثابت با بهترین DWR برابر ۸٪ است. افزایش ۴ برابری اندازه حافظه نهان این فاصله را به ۷٪ می‌رساند و کاهش ۴ برابری اندازه حافظه نهان این فاصله را به ۴٪ کم می‌کند.

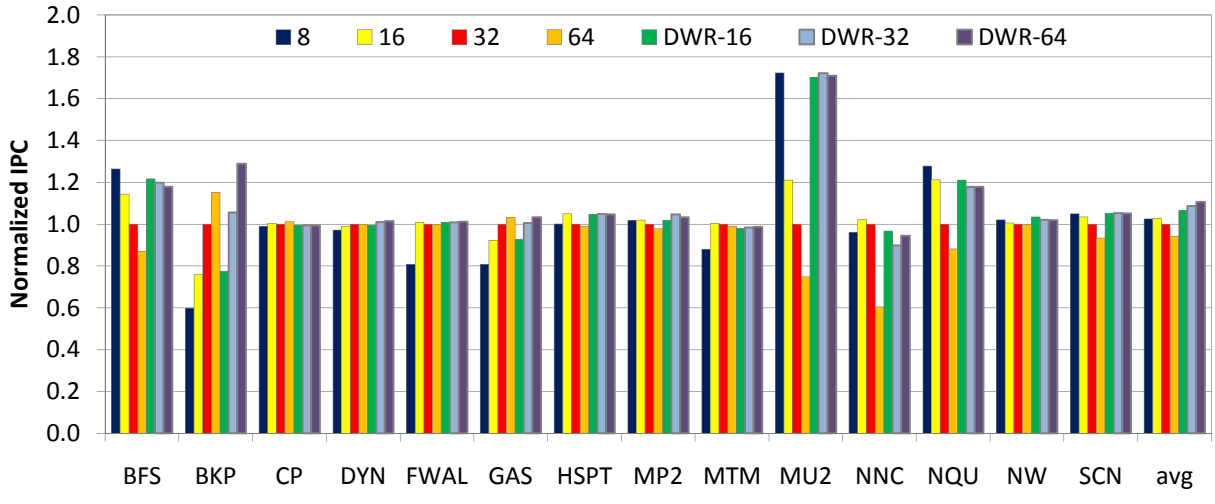
۶-۴-۴-۲ پهنای SIMD

ماشین پایه‌ی این پژوهش از هسته‌های ۸ خطی بهره می‌برد. شکل ۶-۹ روش DWR را با ماشین‌های اندازه وارپ ثابت برای هسته‌های ۱۶ و ۳۲ خطی مقایسه می‌کند. برای هر پهنای SIMD، کوچکترین اندازه وارپ برابر پهنای SIMD است (برای ماشین‌های DWR). اندازه وارپ در هر ماشین با مضربی از پهنای SIMD نشان داده شده است که این عدد برای ماشین‌های اندازه وارپ ثابت همان اندازه وارپ است و برای ماشین‌های DWR اندازه بزرگترین وارپ است. استفاده از SIMD بسیار پهن منجر به افزایش فشار زیر سیستم حافظه می‌شود که ناشی از متوازن نبودن پهنای‌باند اجرا و دسترسی به حافظه است. بنابراین، SIMD

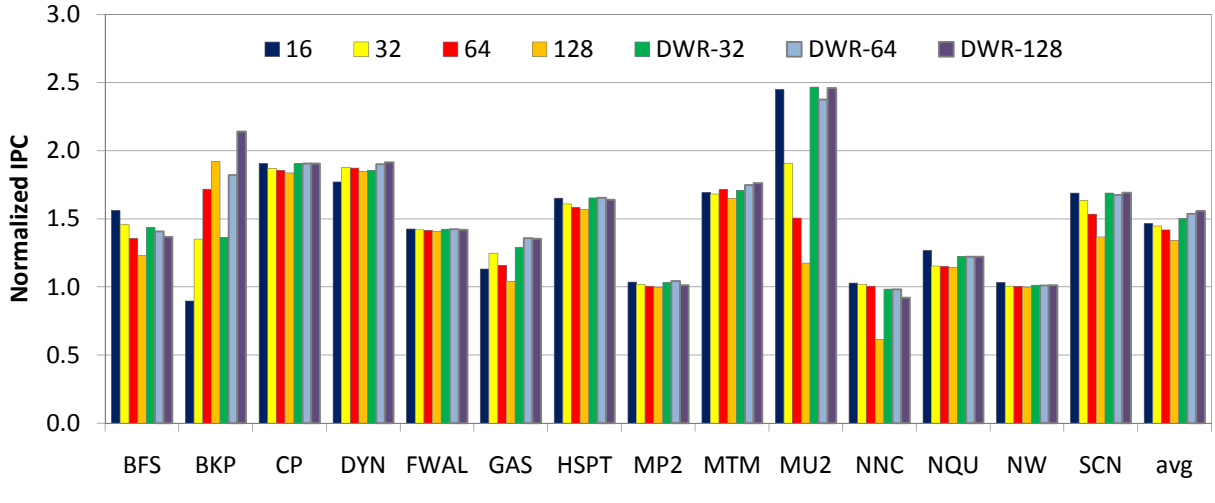


شکل ۶-۸: حساسیت گذردهی به اندازه حافظه نهان برای پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.

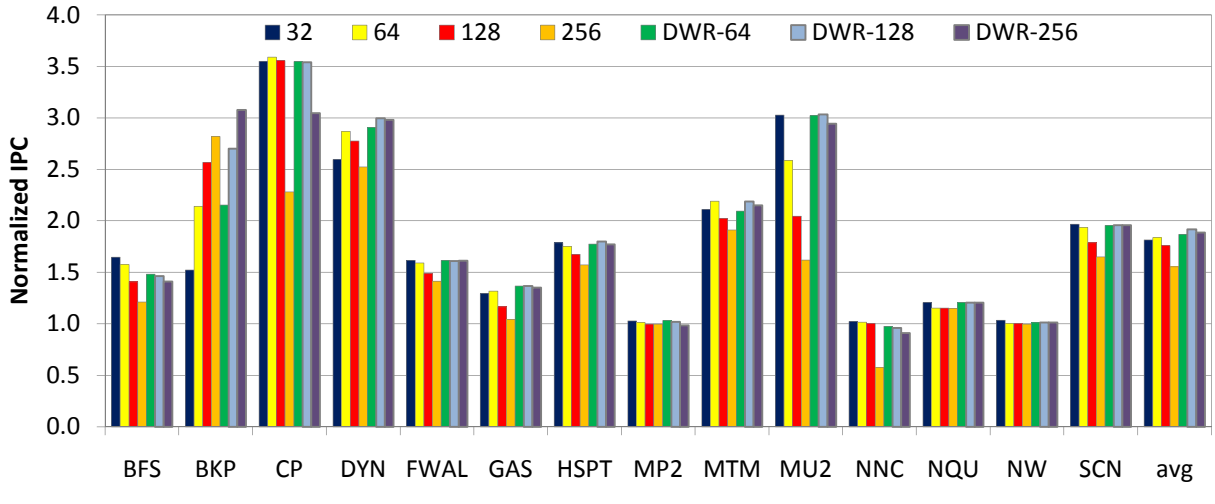
پهن‌تر اثر کاهش واگرایی انشعاب که توسط DWR انجام می‌شود را کم می‌کند زیرا در این شرایط حافظه گلوگاه کارایی است. مقایسه‌ی بهترین DWR با بهترین ماشین اندازه وارپ ثابت نشان می‌دهد که، دو برابر کردن پهنای SIMD هسته‌ها به ۱۶ خط در هر هسته، فاصله گذردهی را ۷٪ کاهش می‌دهد. پهن کردن بیشتر SIMD این فاصله را به ۵٪ می‌رساند. توجه شود که پهن کردن SIMD هسته‌ها، گذردهی را برای NNC و MP بهتر نمی‌کند زیرا NNC تنها ۱۶ نخ در هر بلوک دارد (نهایتا ۱۶ خط از SIMD استفاده می‌شود) و MP بسیار محدود به کارایی سیستم حافظه است.



(الف) ۸ خط در **SIMD**

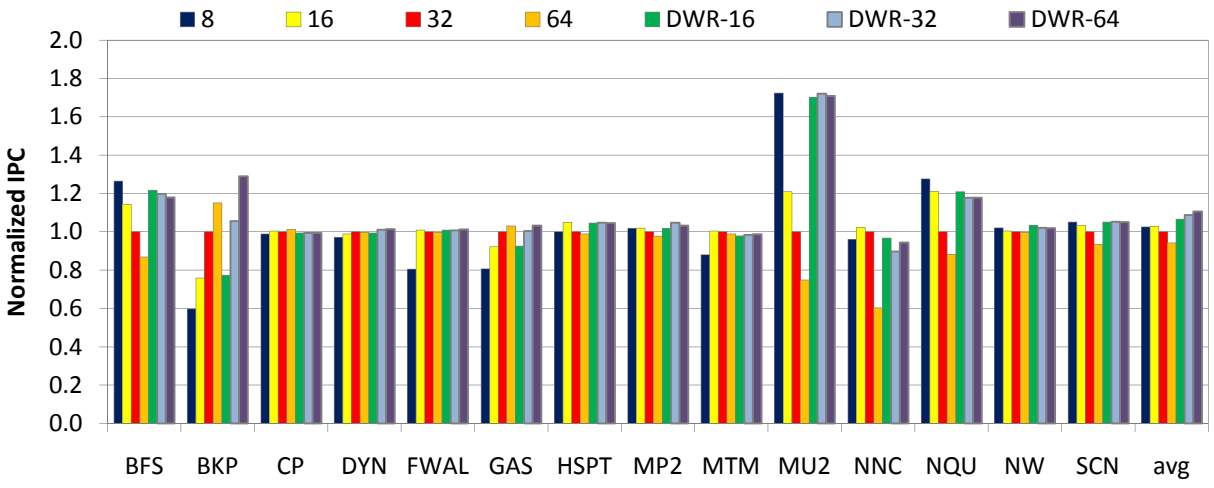


(ب) ۱۶ خط در **SIMD**

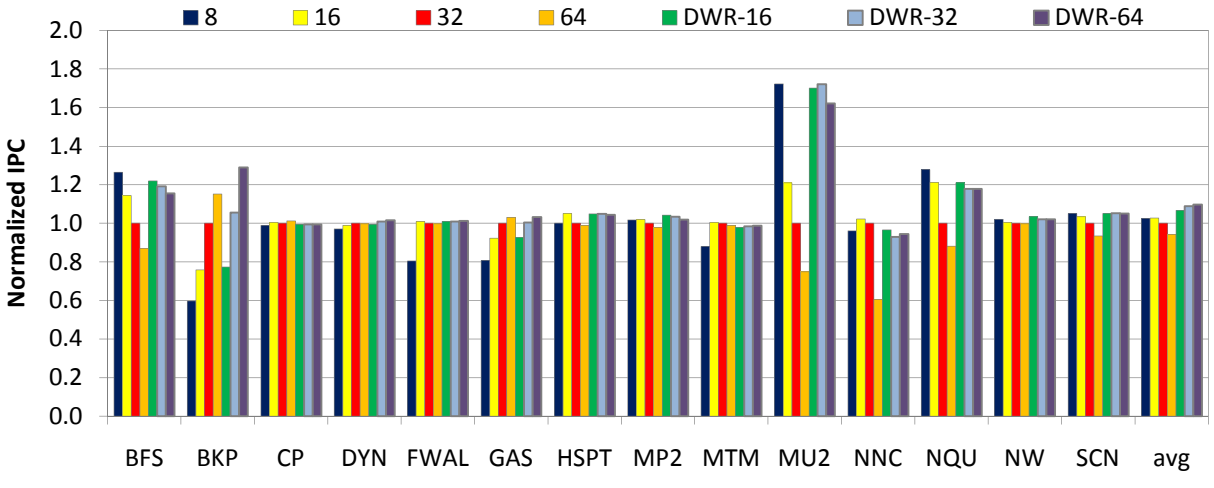


(ج) ۳۲ خط در **SIMD**

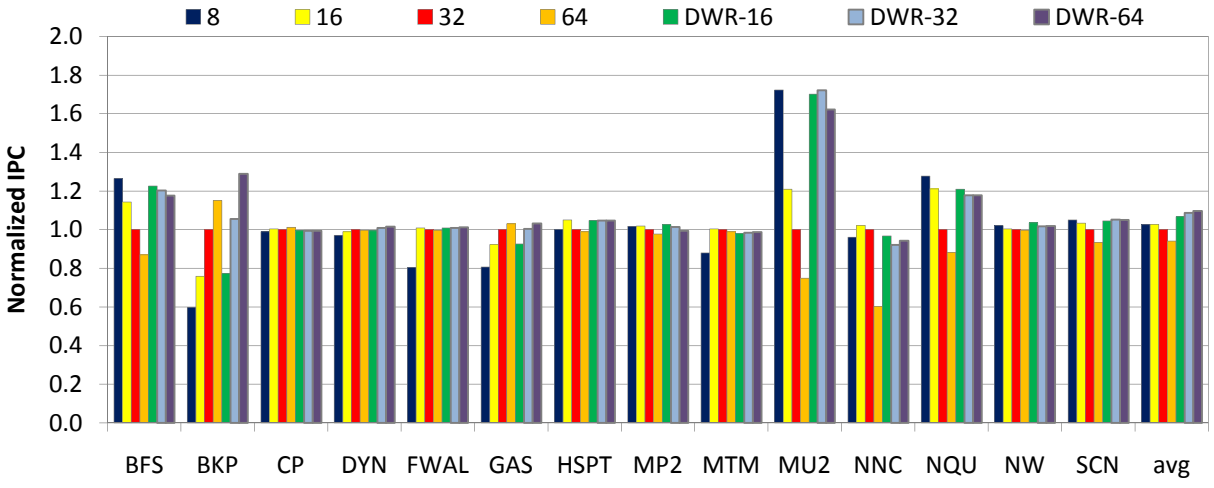
شکل ۶-۹: حساسیت گذردهی به پهنای SIMD برای پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.



(الف) ۳۲ سطر در ۴ مجموعه ۸ راه



(ب) ۱۶ سطر در ۴ مجموعه ۴ راه



(ج) ۸ سطر در ۲ مجموعه ۴ راه

شکل ۶-۱۰: حساسیت گذردهی به اندازه جدول ILT برای پیکربندی‌های متفاوت DWR و ماشین‌هایی با اندازه وارپ ثابت.

۶-۴-۴-۳ اندازه ILT

در این پژوهش، برای ILT حافظه‌ای با ۳۲ سطر در نظر گرفتیم (۴ مجموعه، ۸ راه). همانطوری که در شکل ۶-۱۰ گزارش شده است، جداول ۲ برابر کوچکتر (۱۶ سطر، ۴ مجموعه، ۴ راه) و ۴ برابر کوچکتر (۸ سطر، ۲ مجموعه، ۴ راه) به ۹۹٪ کارایی جدول ۳۲ سطری می‌رسند. در بین محک‌های ارزیابی‌شده، MP بیشترین حساسیت به اندازه ILT را دارد.

۶-۴-۵ بحث

در این بخش، در رابطه با کاربردهای دیگر و پی‌آمدهای استفاده از DWR مطالبی را عنوان می‌کنیم و تا دید بیشتری نسبت به بکارگیری این روش مهیا کنیم.

بارهای کاری کم اثر. اندازه وارپ کارایی محک‌هایی را تحت تاثیر قرار می‌دهد که از واگرایی انشعاب یا واگرایی حافظه تاثیر می‌پذیرند یا آنکه از الحاق دسترسی‌ها به حافظه بهره می‌برند. بنابراین، محک‌هایی که هیچ‌یک از این دو خصوصیت را ندارند (مانند CP و DYN)، به اندازه وارپ حساس نخواهند بود و تبعاً انتظاری از DWR نمی‌رود که کارایی این کاربردها را بهبود دهد.

بهبود وارپ‌های کوچک. DWR می‌تواند به عنوان مکانیزمی برای بهبود کارایی GPUهایی که از وارپ کوچک استفاده می‌کنند، تلقی شود. در بین تمامی پیکربندی‌ها، GPUای که ۸ نخ در وارپ دارد بدترین گذردهی را در اغلب محک‌ها دارد (مانند BKP) زیرا کمترین بهره را از الحاق دسترسی‌ها به حافظ می‌برد. DWR این ماشین را بهبود می‌دهد و بازدهی چشمگیری (تا ۱۱۶٪) دارد. از طرف دیگر، DWR مشابه ماشین وارپ کوچک برای محک‌های محاسباتی (مانند BFS، NU، و NQU) که واگرایی انشعاب غالب‌ترین عامل افت گذردهی در آنها است، خوب عمل می‌کند.

الحاق دسترسی‌ها به حافظه بین وارپ‌ها. DWR می‌تواند به عنوان مکانیزمی برای پیاده‌سازی الحاق بین وارپ‌ها استفاده شود. برای نایل شدن به این هدف، کوچکترین اندازه وارپ برابر با اندازه وارپ ماشین خواهد بود و وارپ‌های بزرگتری در حین اجرای دستورالعمل‌های حافظه ساخته خواهند شد. DWR چندین وارپ را با هم ترکیب می‌کند تا دسترسی‌ها به حافظه‌ی آنها را الحاق کند.

مسائل عملی با وارپ‌های کوچک. جلوی خط‌لوله شامل زمان‌بند وارپ‌ها، موتور واکشی، دیکود کننده دستورالعمل‌ها، و مرحله خواندن ثبات‌ها می‌شود. استفاده از تعداد کمی نخ در وارپ، جلوی خط‌لوله را دستخوش تغییر می‌کند زیرا نیاز به نرخ کلاک بالاتری برای رساندن بارکاری در یک بازه زمانی خواهد بود. افزایش نرخ کلاک می‌تواند باعث افزایش توان مصرفی در جلوی خط‌لوله و محدود کردن پهنای باند واکشی شود. افزون بر این، استفاده از وارپ‌های کوچک می‌تواند باعث بزرگتر شدن جلوی زمان‌بند وارپ‌ها شود زیرا در این حالت باید از بین تعداد بیشتری المان انتخاب کند. این مسئله برای طراحی‌هایی مثل TBC [۱۵]، LWM [۳۸]، و CAPRI [۴۴] نیز وجود دارد. ارزیابی‌های ما نشان می‌دهد که محلیت بسیار بالایی بین دستورالعمل‌های واکشی شده وجود دارد و امکان بهره‌بردن از این محلیت با تکنیک‌هایی مثل فیلتر نهان[3] [۲۵] یا بافر دیکود در حلقه[4] [۱۸] برای کاهش بارکاری جلوی خط‌لوله وجود دارد. در این پژوهش تمرکز ما بر تاثیر اندازه وارپ بر گذردهی بود. تاثیر اندازه وارپ بر مساحت و توان مصرفی بخشی از پژوهش در حال انجام ماست.

رجیستر فایل. اندازه وارپ طراحی و الگوی تخصیص در رجیستر فایل را دستخوش تغییر می‌کند. GPUها تمامی رجیسترهای هم‌نام نخ‌های یک وارپ را در یک ردیف ذخیره می‌کنند [۱۴]. این نوع تخصیص اجازه می‌دهد که با یکبار خواند یک ردیف، یک عملوند برای تمام نخ‌های وارپ خوانده شود. برای حفظ این ساختار برای اندازه وارپ‌های مختلف، تعداد رجیسترهای موجود در یک ردیف (اندازه ردیف) تغییر خواهد کرد. برای وارپ‌ها بزرگتر، اندازه ردیف باید بیشتر باشد تا اجازه خواندن تمام عملوندهای تمام نخ‌ها در یک دسترسی به ردیف مهیا شود و برای وارپ‌های کوچکتر، اندازه ردیف باید کمتر باشد تا از خواندن‌ها اضافی جلوگیری شود.

نسل‌های آینده GPUها. گرایش جاری GPUهای شرکت انویدیا رشدی پایدار در تعداد نخ‌ها، تعداد زمان‌بندهای وارپ، و تعداد واحدهای محاسباتی در هر هسته نشان می‌دهد. DWR مقیاس‌پذیر طراحی شده است و با افزایش تعداد نخ‌ها (وارپ‌ها) در هر هسته همچنان موثر عمل خواهد کرد. همان‌طوری که در این پژوهش سنجیده شد، SIMDهای پهن‌تر مزیت‌های DWR را محدود می‌کنند زیرا اندازه کوچکترین وارپ را افزایش می‌دهند که این افزایش اندازه وارپ باعث تحمیل کردن سربار همگام کردن بیشتری می‌شود. اما

[3] Filter cache
[4] Loop Buffering

توجه شود که پهنای GPUهای امروزی (مانند انویدیا Kepler) بیش از ۱۶ نشده است تا از ریسک طراحی جلوگیری شود [۳۶]. پردازنده‌ی Kepler [۴۰]، ۱۹۲ واحد محاسباتی در هر هسته دارد و واحدهای محاسباتی به ۱۲ گروه با پهنای SIMD برابر با ۱۶ تقسیم شده‌اند. اگرچه ما تنها امکان شبیه‌سازی روی معماری شبیه به Tesla داشته‌ایم، اما معتقدیم که DWR می‌توان برای پردازنده‌هایی شبیه به Kepler [۴۰] و Fermi [۴۷] نیز بهبود گذردهی فراهم آورد.

فصل ۷

نتیجه‌گیری و کارهای آینده

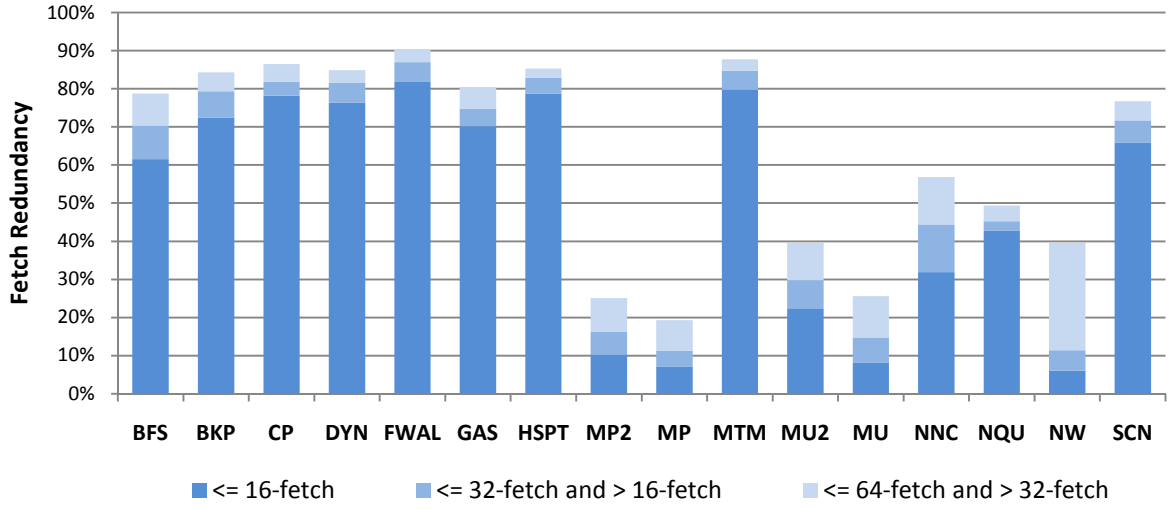
۷-۱ نتیجه‌گیری

در این پژوهش تحلیلی از واگرایی انشعاب در محاسبات همه‌منظوره با پردازنده‌های گرافیکی انجام شد و نشان دادیم که پارامترهایی مثل انتظار روی نقطه بازهمگرایی و غیرفعال شدن نخ‌ها می‌تواند به اندازه‌ی نرخ فعالیت SIMD در گذردهی پردازنده‌های گرافیکی تعیین کننده باشد. مشاهده دیگر این پژوهش تاثیر دوگانه اندازه وارپ بر گذردهی است در زمانی که الحاق دسترسی‌ها به حافظه روی کل وارپ انجام می‌شود. نشان داده شد که در برخی کاربردها، وارپ‌های بزرگ مطلوب هستند و در برخی دیگر وارپ‌های کوچک کارایی بهتری دارند.

مبتنی بر مشاهدات صورت گرفته در این پژوهش، دو تکنیک برای بهبود گذردهی ارائه کردیم. تکنیک اول، CROWN، برای معماری‌هایی پیشنهاد می‌شود که الحاق دسترسی‌ها به حافظه را روی بخشی از وارپ (به پهنای SIMD) انجام می‌دهند. ارزیابی‌ها از تکنیک CROWN نشان داد که این روش در طیف گسترده‌ای از پیکربندی‌های معماری کارایی مناسبی دارد و برای خطلوله‌های کم‌عمق، SIMDهای باریک بهتر عمل می‌کند. تکنیک دوم، DWR، برای معماری‌هایی پیشنهاد می‌شود که الحاق دسترسی‌ها به حافظه را روی کل وارپ انجام می‌دهند. ارزیابی‌ها برای تکنیک DWR نشان می‌دهد که این روش برای SIMDهای باریک بهتر عمل می‌کند.

۷-۲ کارهای آینده

کارآمدی تکنیک‌های ازریابی شده در این پژوهش با لحاظ کردن توان مصرفی یکی از کارهای آینده این پژوهش است. ارزیابی کارآمدی تکنیک‌های پیشنهادی نیازمند شبیه‌سازی کامل (تمامی ماجول‌ها) انرژی



شکل ۷-۱: واکشی تکراری بین وارپ‌ها همروند هر هسته در ۱۶، ۳۲، و ۶۴ سیکل متوالی.

است زیرا تکنیک‌های ارزیابی شده نرخ فعالیت اغلب ماجول‌های سخت‌افزاری در تراشه GPU را تغییر می‌دهند.

روش‌های پیشنهادی CROWN و DWR مبتنی بر استفاده از وارپ‌های کوچک هستند که می‌تواند باعث افزایش نرخ دسترسی به جلوی خطلوله (زمان‌بندی وارپ‌ها، واکشی، و دیکود) شود. بخشی از پژوهش در حال انجام ما در رابطه با بهبود کارآمدی انرژی مصرف‌شده توسط جلوی خطلوله است. در این راستا محلیت زمانی[1] دستورالعمل‌ها را بررسی کرده‌ایم. مشاهدات ما نشان می‌دهد که تعدادی از دستورالعمل، بخش عمده‌ای از واکشی و دیکود دستورالعمل‌ها در بازه‌ی زمانی کوتاهی را شامل می‌شوند. از این محلیت می‌توان استفاده کرد و با تکنیک‌هایی مثل زمان‌بند دوسطحی وارپ‌ها [۵۳]، فیلتر نهان [۲۵]، و بافر دیکود در حلقه [۱۸]، نرخ دسترسی به جلوی خطلوله را کم کرد. در ادامه برخی از نتایج اولیه بدست آمده از مشاهدات خود در این زمینه را ارائه می‌کنیم.

چندنخی عمیق در GPUها نقش مهمی در ایجاد این میزان محلیت دارد. GPUها با بکار گرفتن و درمیان کردن اجرای هزاران نخ در هر هسته، گذردهی را بالا نگه می‌دارند. به جای اتکا کردن بر ILP موجود در هر وارپ، خطلوله با در میان کردن دستورالعمل‌های وارپ‌های متفاوت پر می‌شود. این ساختار خطلوله محلیت زمانی دستورالعمل‌ها را تقویت می‌کند زیرا عموماً دستورالعمل مشابهی در بازه زمانی کوتاه

[1] Temporal Locality

برای وارپ‌های همروند واکشی می‌شود. محلیت زمانی در GPUها از دو حقیقت نشات می‌گیرد: ۱) هسته‌ها اغلب وارپ‌هایی از یک کرنل را اجرا می‌کند و ۲) زمان‌بند، وارپ‌ها را متعادل و منصفانه پیش می‌برد. برای اینکه دید واضحی نسبت به میزان این محلیت این مهیا کنیم و محلیت زمانی دستورالعمل‌ها را نشان دهیم، واکشی‌های تکراری را اندازه‌گیری می‌کنیم. واکشی تکراری، درصد دستورالعمل‌هایی را گزارش می‌کند که بوسیله دیگر وارپ‌ها پیش‌تر واکشی شده است. واکشی تکراری را برای اندازه پنجره دستورالعمل‌های متفاوتی ارزیابی کردیم. شکل ۷-۱ واکشی تکراری برای اندازه پنجره‌های ۱۶ دستورالعمل، ۳۲ دستورالعمل، ۶۴ دستورالعمل، و نامحدود گزارش می‌کند.

به طور متوسط، برای پنجره‌های ۱۶، ۳۲، و ۶۴، به ترتیب تنها ۵، ۶، و ۷ واکشی متمایز دیده می‌شود و اغلب واکشی‌ها تکراری هستند (نتایج برای اندازه وارپ ۳۲ نخی گزارش شده است). محک‌هایی که تعداد زیادی وارپ همروند در هر هسته دارند و واگرایی انشعاب در آنها کم است، بیشترین درصد واکشی تکراری را دارند. دیگر محک‌ها، که تعداد وارپ‌های کمتری دارند (مانند GAS و NW) یا واگرایی انشعاب در آنها فراگیر است (مانند MU، MP، MP2، MU2، و NQU) واکشی تکراری کمتری در بازه زمانی کوتاه (پنجره‌های کوچک) از خود نشان می‌دهند.

به اقسام گوناگونی می‌توان از این محلیت مشاهده شده بهره برد تا کارایی و انرژی پردازنده بهبود یابد. به عنوان مثال می‌توان از یک فیلتر نهان برای کاهش دسترسی‌ها به حافظه نهان دستورالعمل‌ها استفاده کرد تا میزانی از انرژی مصرفی واکشی کمتر شود. استفاده از بافر ردیف دستورالعمل، که آخرین ردیف دسترسی شده از حافظه نهان دستورالعمل‌ها را نهان می‌کند، نیز می‌تواند در کاهش انرژی مصرفی واکشی مفید باشد. همچنین استفاده از بافر دیکود می‌تواند برای کم کردن دسترسی‌ها به واحد واکشی و دیکود بسیار موثر عمل کند.

مراجع

[1] AMD Inc. (2010). ATI Stream Computing: Compute Abstraction Layer (CAL) [online]. Available: *http://www.amddevcentral.com/gpu_assets/ATI_Stream_SDK_CAL_Programming_Guide_v2.0.pdf*

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

[3] E. Blem, M. Sinclair, K. Sankaralingam, "Challenge benchmarks that must be conquered to sustain the GPU revolution," *In Proceedings of the 4th Workshop on Emerging Applications for Manycore Architectur in conjunction with ISCA 2011,* San Jose, CA, 2011.

[4] N. Brunie, S. Collange, G. Diamos, "Simultaneous branch and warp interweaving for sustained GPU performance," *In Proceedings of the 39th International Symposium on Computer Architecture,* Portland, OR, 2012, pp. 49-60.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, L. Sang-Ha, K. Skadron, "Rodinia: a benchmark suite for heterogeneous computing," *In Proceedings of IEEE International Symposium on Workload Characterization,* Austin, TX, 2009, pp. 44-54.

[6] S. Collange, "Stack-less SIMT reconvergence at low cost," unpublished.

[7] S. Collange, D. Defour, Y. Zhang, "Dynamic detection of uniform and affine vectors in GPGPU computations," *In Proceedings of the 2009 International Conference on Parallel Processing,* Delft, 2009, pp. 46-55.

[8] S. Collange, M. Daumas, D. Defour, D. Parello, "Barra, a parallel functional GPGPU simulator," *In Proceedings of 18th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems,* Miami Beach, FL, 2010, pp. 351-360.

[9] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suit," *In Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units,* Pittsburgh, PA, 2010, pp. 63-74.

[10] G. Dasika, A. Sethia, T. Mudge, S. Mahlke, "PEPSC: a power-efficient processor for scientific computing," *In Proceedings of International Conference on Parallel Architectures and Compilation Techniques,* Galvestone, TX, 2011, pp. 101-110.

[11] G. Diamos, A. Kerr, A. Yalamanchili, N. Clark, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," *In Proceedings*

of the 19th international conference on Parallel architectures and compilation techniques, Vienna, Austria, 2010, pp. 353-364.

[12] G. Diamos, B. Ashbaugh, S. Maiyuran, A. Kerr, H. Wu, S. Yalamanchili, "SIMD re-convergence at thread frontiers," *In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture,* Porto Alegre, 2011, pp. 477-488.

[13] W. W. L. Fung, I. Sham, G. Yuan, T. M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow," *In Proceedings of 40th Annual IEEE/ACM International Symposium on Microarchitecture,* Chicago, IL, 2007, pp. 407-420.

[14] W. W. L. Fung, I. Sham, G. Yuan, T.M. Aamodt, "Dynamic Warp Formation: Efficient MIMD Control Flow on SIMD Graphics Hardware," *ACM Transactions on Architecture and Code Optimization (TACO),* vol. 6, no. 7, June 2009.

[15] W. W. L. Fung, T. M. Aamodt, "Thread block compaction for efficient SIMT control flow," *In Proceedings of 17th IEEE International Symposium on High-Performance Computer Architecture,* San Antonio, TX, 2011, pp. 25-36.

[16] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," *In Proceedings of the 38th Annual International Symposium on Computer Architecture,* San Jose, CA, 2011, pp. 235-246.

[17] A. Gharaibeh, M. Ripeanu, "Size matters: space/time tradeoffs to improve GPGPU applications performance," *In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis,* New Orleans, LA, 2010, pp. 1-12.

[18] M. Hiraki, R. S. Bajwa, H. Kojima, D. J. Gorny, K. Nitta, A. Shri, "Stage-skip pipeline: a low power processor architecture using a decoded instruction buffer," *International Symposium on Low Power Electronics and Design,* Monterey, CA, 1996, pp. 353-358.

[19] S. Hong, H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *In Proceedings of the 36th Annual International Symposium on Computer Architecture,* Austin, TX, 2009, pp. 152-163.

[20] S. Hong, H. Kim, "An integrated GPU power and performance model," *In Proceedings of the 37th Annual International Symposium on Computer Architecture,* Saint-Malo, France, 2010, 280-289.

[21] W. Jia, K. A. Shaw, M. Martonosi, "Stargazer: automated regression-based GPU design space exploration," *In Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software,* New Brunswick, NJ, 2012, pp. 2-13.

[22] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, D. Glasco. "GPUs and the future of parallel computing," *IEEE Micro,* vol. 31, pp. 7-17, Sept.-Oct. 2011.

[23] A. Kerr, G. Diamos, S. Yalamanchili, “A characterization and analysis of PTX kernels." *In Proceedings of IEEE International Symposium on Workload Characterization,* Austin, TX, 2009, pp. 3-12.

[24] Khronos Group. (2012). OpenCL - The open standard for parallel programming of heterogeneous systems. Available: http://www.khronos.org/opencl/

[25] J. Kin, M. Gupta, W. H. Mangione-Smith, “The filter cache: an energy efficient memory structure,” *In Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture,* Research Triangle Park, NC, 1997, pp. 184-193.

[26] N. B. Lakshminarayana, H. Kim, “Effect of instruction fetch and memory scheduling on GPU performance,” *Workshop on Language, Compiler, and Architecture Support for GPGPU, in conjunction with HPCA/PPoPP,* Bangalore, India, 2010.

[27] A. Lashgar, A. Baniasadi, “Performance in GPU architectures: potentials and distances,” *9th Annual Workshop on Duplicating, Deconstructing, and Debunking in conjunction with ISCA 2011,* San Jose, CA, 2011.

[28] A. Lashgar, A. Baniasadi, A. Khonsari, “Dynamic warp resizing: analysis and benefits in high-performance SIMT,” *In Proceedings of the 30th IEEE International Conference on Computer Design,* Montreal, Canada, 2012, pp 502-503.

[29] C. Lattner, V. Adve, “LLVM: a compilation framework for lifelong program analysis & transformation,” *In Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization (CGO '04),* San Jose, CA, 2004, pp. 75-88.

[30] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, P. Dubey, “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU,” *In Proceedings of the 37th annual international symposium on Computer architecture,* Saint-Malo, France, 2010, pp. 451-460.

[31] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, K. Asanović, “Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators,” *In Proceedings of the 38th Annual International Symposium on Computer Architecture,* San Jose, CA, 2011, pp. 129-140.

[32] J. Lee, P. P. Ajgaonkar, N. S. Kim, “Analyzing throughput of GPGPUs exploiting within-die core-to-core frequency variation,” *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software,* Austin, TX, 2011, pp. 237-246.

[33] E. Lindholm, B. W. Coon, J. Wierzbicki, R. J. Stoll, S. F. Oberman, “Credit-based streaming multiprocessor warp scheduling,” U.S. Patent 0 072 244, Mar 24, 2011.

[34] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, “NVIDIA tesla: a unified graphics and computing architecture,” *IEEE Micro,* vol. 28, pp. 39-55, March-April 2008.

[35] J. Meng, D. Tarjan, K. Skadron, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," *In Proceedings of the 37th annual international symposium on Computer architecture,* Saint-Malo, France, 2010, pp. 235-246.

[36] J. Meng, J. W. Sheaffer, K. Skadron, "Robust SIMD: Dynamically Adapted SIMD Width and Multi-Threading Depth," *In Proceedings of the IEEE International Parallel & Distributed Processing Symposium,* Shanghai, China, 2012, pp. 107-118.

[37] S. S. Muchnick, "Control-Flow Analysis," in Advanced Compiler Design and Implementation, Morgan Kaufmann, 1997, ch. 7, sec. 3, pp. 181-190.

[38] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, 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,* Porto Alegre, 2011, pp. 308-317.

[39] NVIDIA Corp. (2011). CUDA C Programming Guide [online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[40] NVIDIA Corp. (2012). Kepler GK110 Architecture [online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[41] NVIDIA Corp. (2008). NVIDIA CUDA SDK 2.3 [online]. Available: https://developer.nvidia.com/cuda-toolkit-23-downloads

[42] NVIDIA Corp. (2011). PTX: Parallel Thread Execution ISA [online]. Available: http://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[43] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary. High-Performance Linpack [online]. Available: http://www.netlib.org/benchmark/hpl/

[44] M. Rhu, M. Erez, "CAPRI: prediction of compaction-adequacy for handling control-divergence in GPGPU architectures," *In Proceedings of the 39th International Symposium on Computer Architecture,* Portland, OR, 2012, pp. 61-71.

[45] J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, W. M. W. Hwu, "Parboil: a revised benchmark suite for scientific and commercial throughput computing," *IMPACT Technical Report, 2012*.

[46] Top500 Supercomputing Sites. (November 2012). Available: http://top500.org/lists/2012/11/

[47] C. M. Wittenbrink, E. Kilgariff, A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, pp. 50-59, March-April 2011.

[48] H. Wong, M. M. Papadopoulou, M. Sadooghi-Alvandi, A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," *In Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software,* White Plains, NY, 2010, pp. 235-246.

[49] H. Wu, G. Diamos, J. Wang, S. Li, S. Yalamanchili, "Characterization and transformation of unstructured control flow in bulk synchronous GPU applications,"

International Journal of High Performance Computing Applications, vol. 26, no. 2, pp. 170-185, May 2012.

[50] G. L. Yuan, A. Bakhoda, T. M. Aamodt, "Complexity effective memory access scheduling for many-core accelerator architectures," *In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture,* New York, NY, 2009, pp. 34-44.

[51] E. Z. Zhang, Y. Jiang, Z. Guo, K. Tian, X. Shen, "On-the-fly elimination of dynamic irregularities for GPU computing," *In Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems,* New York, NY, USA, 2011, pp. 369-380.

[52] N. Muralimanohar, R. Balasubramonian, N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," *In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture,* Chicago, IL, 2007, pp. 3-14.

[53] W. J. Dally, S. Keckler, D. Tarjan, J. E. Lindholm, M. Gebhart, D. Johnson. "Two-level scheduler for multi-threaded processing," U.S. Patent 0 079 503, March 29, 2012.

واژه‌نامه فارسی به انگلیسی

اعتبار ..... Validity

آفلاین ..... Offline

آماده ..... Ready

اندازه وارپ ..... Warp Size

انشعاب شرطی ..... Conditional branch

ایشو ..... Issue

بارکاری ..... Workload

بارگذاری ..... Load

بافت ..... Texture

بافر ..... Buffer

بسیار هسته‌ای ..... Many-core

بهره‌برداری ..... Utilization

بی‌اعتبار کردن ..... Invalidate

بیکار ..... Idle

بی‌نظمی ..... Irregular

پردازنده کمکی ..... Co-processor

پروسس ..... Process

پست‌دومیناتور بلافاصله ..... Immediate postdominator

پسنویسی ..... Writeback

پوشاندن ..... Hiding

پیچیدگی ..... Complexity

تاخیر ..... Latency

تخصیص ..... Allocate

تراشه ..... Chip

ترکیب ..... Merge

تودرتو ..... Nested

گرید ..... Grid

ثبات ..... Register

جاروب ..... Sweep

چالش ..... Challenge

چندنخی ..... Multi-threading

الحاق ..... Coalescing

خاص‌منظوره ..... Fixed-function

خط ..... Lane

خط‌لوله ..... Pipeline

در میان کردن ..... Interleaving

دسترسی ..... Access

ذخیره ..... Store

رد ..... Trace

ردیف ..... Row

رزرو کردن ..... Reserve

زمان‌بند ..... Scheduler

زمان‌بندی ..... Scheduling

سربار ..... Overhead

شبکه‌ی برتراشه ..... Network-on-Chip

شتاب‌دهنده ..... Accelerator

شناسه ..... Identifier (ID)

غیرفعال ..... Inactive

فعال ..... Active

فقدان ..... Miss

کاربرد ..... Application

گذردهی ..... Throughput

گلوگاه ..... Bottleneck

لینک ..... Link

متقارن ..... Symmetric

متوازن ..... Balanced

محاسبات همه‌منظوره ..... General-purpose computation

محافظه کارانه ..... Conservative

محلیت ..... Locality

مساحت ..... Area

مسیریاب ..... Router

مصالحه ..... Tradeoff

معتبر ..... Valid

معماری ..... Architecture

نامتقارن ..... Asymmetric

نحو ..... Syntax

نخ ..... Thread

نرخ ..... Rate

نرخ فعالیت SIMD ..... Lane Activity

نظم ..... Regular

نهان ..... Cache

همپوشانی دادن ..... Overlapping

همگام ..... Synchronized

همگام کردن ..... Synchronize

همگرا ..... Re-converge

همگرایی ..... Re-convergence

همه‌منظوره ..... General-Purpose

وارپ ..... Warp

واژگان ..... Terminology

واگرا ..... Diverge

واگرایی ..... Divergence

واگرایی انشعاب ..... Branch divergence

واگرایی حافظه ..... Memory divergence

واژه‌نامه انگلیسی به فارسی

Accelerator	شتاب‌دهنده
Access	دسترسی
Active	فعال
Allocate	تخصیص
Application	کاربرد
Architecture	معماری
Area	مساحت
Asymmetric	نامتقارن
Balanced	متوازن
Bottleneck	گلوگاه
Branch divergence	واگرایی انشعاب
Buffer	بافر
Cache	نهان
Challenge	چالش
Chip	تراشه
Coalescing	الحاق
Complexity	پیچیدگی
Conditional branch	انشعاب شرطی
Conservative	محافظه کارانه
Co-processor	پردازنده کمکی
Diverge	واگرا
Divergence	واگرایی
Fixed-function	خاص‌منظوره
General-purpose computation	محاسبات همه‌منظوره
General-Purpose	همه‌منظوره
Grid	گرید
Hiding	پوشاندن
Identifier (ID)	شناسه
Idle	بیکار
Immediate postdominator	پست‌دومیناتور بلافاصله

Inactive ..... غیرفعال

Interleaving ..... در میان کردن

Invalidate ..... بی‌اعتبار کردن

Irregular ..... بی‌نظمی

Issue ..... ایشو

Lane Activity ..... نرخ فعالیت SIMD

Lane ..... خط

Latency ..... تاخیر

Link ..... لینک

Load ..... بارگذاری

Locality ..... محلیت

Many-core ..... بسیار هسته‌ای

Memory divergence ..... واگرایی حافظه

Merge ..... ترکیب

Miss ..... فقدان

Multi-threading ..... چندنخی

Nested ..... تودرتو

Network-on-Chip ..... شبکه‌ی برتراشه

Offline ..... آفلاین

Overhead ..... سربار

Overlapping ..... همپوشانی دادن

Pipeline ..... خط‌لوله

Process ..... پروسس

Rate ..... نرخ

Ready ..... آماده

Re-converge ..... همگرا

Re-convergence ..... همگرایی

Register ..... ثبات

Regular ..... نظم

Reserve ..... رزرو کردن

Router ..... مسیریاب

Row ..... ردیف

Scheduler ..... زمان‌بند

Scheduling ..... زمان‌بندی

Store ..... ذخیره

Sweep ..... جاروب

Symmetric ..... متقارن

Synchronize ..... همگام کردن

Synchronized ..... همگام

Syntax ..... نحو

Terminology ..... واژگان

Texture ..... بافت

Thread ..... نخ

Throughput ..... گذردهی

Trace ..... رد

Tradeoff ..... مصالحه

Utilization ..... بهره‌برداری

Valid ..... معتبر

Validity ..... اعتبار

Warp Size ..... اندازه وارپ

Warp ..... وارپ

Workload ..... بارکاری

Writeback ..... پسنویسی

Abstract

Today supercomputers employ throughput-oriented GPUs as a computation accelerator. In these processors, thousands of threads per core are interleaved to hide the memory latency. Each core groups tens of threads into a warp and executes the threads of a warp at the same pace in order to utilize the wide SIMD. Upon executing conditional branch instruction, threads of a warp can diverge into different paths. Conventional mechanism of branch divergence management reduces SIMD utilization significantly. In this study we evaluate the branch divergence and show that the processor throughput (SIMD utilization) is a tradeoff between the thread-level parallelism and the SIMD lane activity. Moreover, we show that the warp size can impact the processor performance. A given architecture may perform faster under small or large warps, depending on the workload behavior. Based on these observations, we propose two techniques to improve the GPU performance. In the first technique, known as CROWN, a novel control flow mechanism has been presented that regroups the diverged threads into new warps and rejoins the threads at the re-convergence point dynamically. The CROWN's goal is to improve both the thread-level parallelism and SIMD lane activity. Our evaluation shows that the CROWN improves performance up to 2.3X while imposing the hardware overhead of ~4%. The second technique, DWR, schedules the threads in small warps and dynamically synchronizes them to execute memory instructions using large warps. The DWR goal is to achieve the benefits of both small and large warp under the unified design. Our evaluation shows that the DWR improves the performance up to 70% while imposing less than 1% area overhead.

Keywords: Processor Architecture, Hardware Accelerator, GPU, SIMD Utilization, Multithreading, Thread Grouping,

University of Tehran

School of Electrical and Computer Engineering

Improving GPU Performance with Improved SIMD Efficiency

By

Ahmad Lashgar

Primary Supervisor: **Dr. Ahmad Khonsari**

Secondary Supervisor: **Dr. Amirali Baniasadi**

A thesis submitted to the Graduate Studies Office
In partial fulfillment of the requirements
For the degree of MS
in
Computer Engineering

November 2012