**دانشگاه صنعتی امیرکبیر**
**(پلی تکنیک تهران)**
**دانشکده مهندسی برق**

**گزارش سمینار در میکرو دو**
**گرایش الکترونیک دیجیتال**

# معماری پردازنده گرافیکی پاسکال

**نگارش**
**علی زمانی**

**استاد درس**
**دکتر سید احمد معتمدی**

**بهمن ماه 1397**

# فهرست مطالب

صفحه

# فهرست اشکال

صفحه

# فصل اول

## مقدمه

## مقدمه

پردازنده های گرافیکی امروزه در زمینه های گوناگونی از قبیل بازی ها، نرم افزار های گرافیکی همچون فوتوشـــاپ ، corel و ... مورد مصـــرف قرار میگیرند. با روی کار آمدن الگوریتم های شـــبکه ژرف و هوش م صنوعی در چندین سال اخیر و نیاز محا سباتی زیاد آنها، پردازنده های گرافیکی راه خود را در این بین باز کرده و سر دم دار عرصه سخت افزار های قدرتمد برای پردازش های موازی در حیطه شبکه های چند لایه ژرف شده اند.

شرکت NVidia با عرضه معماری جدیدی تحت عنوان پاسکال این امکان را به برنامه نویسان و محققان داده اســـت که برنامه های شـــبکه عصـــبی و یادگیری ژرف خود را در کمتر از دقیقه به اجرا در آورند، الگوریتم هایی که اجرای آنها بر روی ســـخت افزار های ســـنتی ممکن بود تا روز ها به طول انجامد. این شرکت با همکاری های و سیع با کمپانی های نرم افزاری و اینترنتی دنیا از قبیل فیسبوک و نتفلیکس و گوگل مسـیر را برای بسـیاری از الگوریتم های یادگیری ژرف هموار سـاخته اسـت. همچنین بسـیاری از کمپانی های ما شین سازی دنیا نظیر آوودی و بنز با بهره گیری از این پردازنده های گرافیکی انقلابی در پدیده ماشین های بدون راننده کرده اند.

# فصل دوم

# معماری پردازنده‌های گرافیکی

# پردازنده گرافیکی در یک نگاه

انسان‌ها قادر به درک سه بعد هستند و از دنیای اطرافشان دیدی سه‌بعدی دارند. همین ویژگی انسان‌ها سبب شده است که از سه دهه پیش تاکنون تلاش‌های بسیاری برای شبیه سازی دنیای سه‌بعدی خارجی بر روی صفحه‌ی نمایشگر رایانه‌ها که دارای دو بعد است صورت پذیرد. این تلاش‌ها که متمرکز بر نمایش سه‌بعدی یک تصویر دوبعدی است، به محاسبات بسیار سنگینی نیاز دارد این محاسبات در صورتی‌که قرار باشد باکیفیت بالایی انجام شوند، از عهده‌ی CPU[1]ها که پردازنده‌هایی قدرتمند و همه‌منظوره هستند نیز خارج است. البته ممکن است با توجه به توان CPU و نوع کاری که مدنظر است، امکان واگذار کردن پردازش گرافیکی به CPU ممکن باشد اما این کار به قیمت بازداشتن CPU از رسیدن به کارهای اصلی‌اش تمام خواهد شد. به همین دلایل با مرور زمان تصمیم بر این شده است که پردازنده‌ای جداگانه برای پردازش‌های گرافیکی اختصاص پیدا کند.برخی از افراد این پردازنده، یعنی واحد پردازش گرافیکی[2] را با کارت گرافیک[3] یکی می‌دانند درحالی‌که بین این دو قسمت تفاوت‌های بارزی وجود دارد. کارت گرافیک قطعه‌ای است که به کمک آن رایانه قادر می‌گردد تصویر را تولید کرده و برای نمایش در اختیار نمایشگر قرار دهد. یکی از قسمت‌های کارت گرافیک واحد پردازش گرافیکی است. در این گزارش به واحد پردازش گرافیکی به‌اختصار پردازنده‌ی گرافیکی گفته می‌شود. پردازنده‌ی گرافیکی مهم‌ترین قسمت یک کارت گرافیک است که مسئول اجرای پردازش‌های لازم برای نمایش تصویر است. در این فصل ما به‌صورت خلاصه در مورد کارت‌های گرافیک توضیح خواهیم داد، سپس بحث پردازش موازی با کمک پردازنده‌های گرافیکی را با توضیح معماری CUDA[4] آغاز می‌کنیم و سپس بحث را با شرح معماری پردازنده‌های گرافیکی ساخت شرکت NVIDIA ادامه خواهیم داد.

## 2.1 کارت‌های گرافیک

نخستین کارت گرافیک در سال 1960 میلادی توسط شرکت IBM ساخته شد. این کارت گرافیک که همراه رایانه‌های IBM PC فروخته می شد دارای 4 کیلوبایت حافظه بود و فقط قادر بود رنگ سبز را نشان دهد که از آن برای نمایش کارکترها استفاده می شد. هرچند نخستین کارت گرافیک دارای قدرت پرداز شی بالایی نبود اما همین کارت ساده نویدبخش تحول بزرگی در صنعت پردازش الکترونیک بود. با توجه به اینکه تصویر دارای جذابیت‌های خاص خودش است، هر تغییری که در ساختار کارت‌های گرافیک ایجاد و موجب بهبود تصویر نهایی می‌شد از سوی کاربران با استقبال فراوان روبه‌رو می‌شد. از طرف دیگر با رشد تدریجی کارت‌های گرافیکی و

[1]Central Processing Unit
[2] Graphics Processing Unit
[3]Graphic Card
[4]Compute Unified Device Architecture

افزایش قدرت آن‌ها صنعت بازی‌های رایانه‌ای شروع به کارکرد. همچنین وجود کاربران متعدد باعث تضمین سود نا شی از فعالیت در بازار تو سعه کارت‌های گرافیکی شد. بدین سان صنعت تو سعه کارت‌های گرافیکی با سرعتی فزاینده شروع به رشد کرد. وظیفه‌ی کارت گرافیک این است که سیگنال‌های دیجیتال تولیدشده توسط پردازنده را دریافت کرده و آن‌ها را تبدیل به سیگنال‌های آنالوگ یا دیجیتال قابل‌نمایش توسط نمایشگر بکند. شایان‌ذکر است که این کار باید در زمان بسیار کمی انجام شود به‌گونه‌ای که تغییرات برای چشم انسان آزاردهنده نباشد.

هر کارت گرافیک دارای واحدهای مختلفی است که هرکدام برای کار خاصی در نظر گرفته شده‌اند. این واحدها در کارت‌های مختلف ممکن است تغییر کنند. ازآنجایی‌که بحث در مورد طراحی و جزئیات کارت‌های گرافیکی از موضوع بحث این گزارش خارج است، در ادامه برخی از قسمت‌های مهم به‌صورت خلاصه توضیح داده خواهند شد.

**پردازنده‌ی گرافیکی:** این واحد مهم‌ترین قسمت کارت گرافیک است که عملاً مسئولیت پردازش تصاویر را بر عهده دارد. بحث ما در این کتاب روی این واحد متمرکز است و توضیح خواهیم داد که چگونه می‌توان از این واحد برای پردازش موازی استفاده کرد.

**حافظه:** حافظه برای ذخیره‌سازی اطلاعات مربوط به تصاویر و کار بر روی آن‌هاست. در کارت‌های گرافیک جدیدتر از حافظه‌هایی با ظرفیت زیاد استفاده می شود زیرا در بسیاری از مسائل پردازش موازی با ابعاد وسیع حجم حافظه تأثیر به سزایی بر روی سرعت حل مسئله خواهد داشت.

**ادوات جانبی:** در هر کارت گرافیک یک سری ادوات جانبی نیز وجود دارد. برای مثال می‌توان به واسط ارتباط با برد اصلی[1] اشاره کرد؛ این واسط می‌تواند PCI[2]، PCIe[3]، AGP[4] و ... باشد در حال حاضر[5] بیشتر پردازنده‌های گرافیکی از PCI e استفاده می‌کنند. درگاه‌های[6] خروجی تصاویر نیز از دیگر ادوات جانبی هستند. این درگاه‌ها که برای انتقال اطلاعات بر روی نمایشگر ساخته‌شده‌اند ممکن است دارای استاندارد DIV[7] یا VGA[8] باشند. کارت‌های گرافیکی دارای یک قسمت بایاس هم هستند که عملاً برنامه‌ای است که کارت گرافیک به کمک آن با CPU ارتباط برقرار می‌کند و دستورهای CPU را اجرا می‌کند. ازآنجایی‌که پردازنده‌های گرافیکی گرمای زیادی تولید می‌کند کارت‌های گرافیکی معمولاً دارای چندین مبدل حرارتی و پنکه برای تعدیل دما هستند.در هنگام نصب

[1]Mainboard
2Peripheral Component Interconnect
3PCI Express
4Accelerated Graphic Port
5سال 2013
6Port
[7]Digital Video Interface
[8]Video Graphics Array

کارت گرافیک نیز باید مسائل مربوط به خنک شدن در نظر گرفته شود. از دیگر مزیت‌های موجود در کارت‌های گرافیک می‌توان به خطوط لوله بازنمایی[1] ، واحدهای بافت و سایه زنی اشاره کرد.

همان‌طور که پیش‌تر ذکر شد با وارد شدن بازی‌های رایانه‌ای سرعت توسعه کارت‌های گرافیک بیشتر شد زیرا با ارائه هر کارت گرافیک قدرتمندتر بازی‌های جذاب‌تری قابل ساخت بود و هر بازی جدید خود پردازنده‌ی گرافیکی قوی‌تری را طلب می‌کرد. این حلقه‌ی تشدید شونده و علاقه‌ی روزافزون قشر وسیعی از کاربران به بازی‌های رایانه‌ای از یک سو و جذابیت‌های بصری رایانه برای تمامی کاربران از سوی دیگر باعث شد که روزبه‌روز پردازنده‌های گرافیکی قدرتمندتری تولید شود و کاربران به‌راحتی آن‌ها را خریداری کنند. دراین‌بین برخی از کاربران به این فکر افتادند که از قدرت بالای پردازنده‌های گرافیکی برای کارهای مفیدتری! استفاده کنند. این نقطه‌ی آغازی بر ا ستفاده از پردازنده‌های گرافیکی برای کارهای عمومی و علمی بود. واقعیت این ا ست که در حال حا ضر بسیاری از رایانه‌ها دارای پردازنده‌های گرافیکی بسیار قدرتمندی هستند. این پردازنده‌ها تا زمانی که رایانه درگیر انجام کارهای پردازش تصاویر پیچیده است توانایی خارق‌العاده‌ای به آن می‌دهند اما به‌صورت کلی اکثر رایانه‌ها وقت بسیار کمی را برای پردازش تصاویر اختصاص می‌دهند و در سایر زمان‌ها پردازنده گرافیکی بیکار مانده، انرژی مصرف کرده و آن را تبدیل به گرما می‌کند. با در نظر گرفتن این نکات استفاده از پردازنده گرافیکی برای سایر مقاصد پردازشی از هر جنبه مفید به نظر می‌رسد هرچند که این کار در ابتدا کار دشواری بود زیرا پردازنده‌های گرافیکی اولیه دارای دو واحد پردازش پیکسلی[2] و پردازش رأسی[3] بودند. واحد پردازش رأسی مسئول انجام محاسبات رأسی بود. برای مثال در واحد پردازش رأسی باید مختصات یک تصویر سه‌بعدی در یک فضای دوبعدی (صفحه‌ی مانیتور) به دست آید. واحد پردازش پیکسلی هم مسئول مشخص کردن رنگ بخش‌های مختلف تصویر بود. با توجه به این نکات در آغاز کار برای پردازش موازی با پردازنده‌ی گرافیکی می‌بایست برنامه‌ها به فرم مسائل پردازش تصویر دربیایند و سپس برای پردازش به پردازنده‌ی گرافیکی داده شوند. این کار نه‌تنها دشوار بود بلکه برای تمامی مسائل هم عملی نبود. به تدریج شرکت‌های سازنده‌ی پردازنده‌های گرافیکی سعی کردند که پردازنده‌ی خاص‌منظوره‌ی خود را از خاص منظور بودن درآورده و آن را برای اجرای عملیات پردازش موازی بهینه نمایند. برای نیل به این هدف بردا شتن دو گام لازم بود. گام نخ ست تغییر معماری سخت‌افزار و ارائه معماری منا سب پردازش

[1] Rendering pipeline
[2] Pixel Fragment Processor
[3] Vertex Processor

موازی بود و گام دیگر توسعه‌ی زبان‌های برنامه‌نویسی مناسب برای فراهم کردن امکان کار با پردازنده‌ی گرافیکی بود. شرکت NVIDIA این دو گام را برداشت و یک معماری تحت عنوان CUDA[1] به وجود آورد[۲].

## 2.2 معماری CUDA[3]

کودا که نخستین بار در سال 2006 هم‌زمان با ارائه‌ی GeForce 8800 رونمایی شد، یک بستر نرم‌افزاری[۴] برای توسعه‌ی برنامه‌های موازی بر روی برخی[۵] پردازنده‌های گرافیکی ساخت شرکت NVIDIA را فراهم می‌کند. با استفاده از این بستر نرم‌افزاری، می‌توان روی پردازنده‌های گرافیکی دقیقاً مانند CPU برنامه‌نویسی کرد. به این استفاده از پردازنده‌های گرافیکی برای حل مسائل عمومی GPGPU[۶] گفته‌شده که گاهی اوقات به‌اختصار به‌صورت GP[2]U نوشته می‌شود. امروزه NVIDIA به‌جای GPGPU از اصطلاح GPU Computing استفاده می‌کند. کودا به‌صورت یک افزونه برای زبان‌های برنامه‌نویسی استاندارد C، C++ و Fortran وجود دارد. شایان‌ذکر است که این زبان‌ها تنها راه استفاده از پردازنده‌های گرافیکی نیست بلکه امکان کار با پردازنده‌های گرافیکی از طرق دیگری نیز مانند استفاده از OpenCL و یا استفاده از جعبه‌ابزارهای متلب و LabVIEW میسر است. مدلی که کودا برای پردازش ارائه می‌کند به‌صورت زیر است.

1. حافظه‌ی موردنیاز بر روی حافظه‌ی پردازنده‌ی گرافیکی رزرو می‌شود.
2. اطلاعات از حافظه‌ی اصلی یا ورودی‌ها برداشته‌شده و به حافظه‌ی پردازنده‌ی گرافیکی انتقال داده می‌شود.
3. CPU دستوراتی را که باید اجرا شوند برای اجرا به پردازنده‌ی گرافیکی می‌فرستد[۷].
4. پردازنده‌ی گرافیکی دستورات را به‌صورت موازی بر روی‌داده‌هایی که در مرحله‌ی یک دریافت کرده است اجرا می‌کند.

---

[1] Compute Unified Device Architecture

[2] شایان ذکر است که هرچند CUDA مخفف عبارت Compute Unified Device Architecture است اما این عبارت مخفف فقط قبلاً به کار گرفته می‌شد. در حال حاضر شرکت NVIDIA این عبارت مخفف را حذف کرده است و فقط از CUDA به عنوان یک نشانه (Brand) استفاده می‌کند، نشانه‌ای که امروزه (2013) خیلی معروف شده است.

[3] برای راحتی از اینجا به بعد واژه‌ی CUDA را به صورت «کودا» خواهیم نوشت.

[4] Platform

[5] پردازنده‌های گرافیکی قدیمی از این بستر پشتیبانی نمی‌کنند. برای اینکه بدانید که آیا یک پردازنده‌ی گرافیکی از این بستر پشتیبانی می‌کند یا خیر به دفترچه راهنما یا سایت شرکت NVIDIA مراجعه نمایید.

[6] General-Purpose computing on Graphics Processing Units

[7] Run kernel

5. اطلاعات پس از هم‌زمان‌سازی CPU با پردازنده‌ی گرافیکی از حافظه‌ی پردازنده‌ی گرافیکی برداشته‌شده و به مقصد (حافظه‌ی اصلی یا خروجی) انتقال پیدا می‌کند.

6. حافظه‌ای که در پردازنده‌ی گرافیکی رزرو شده است آزاد می‌گردد.

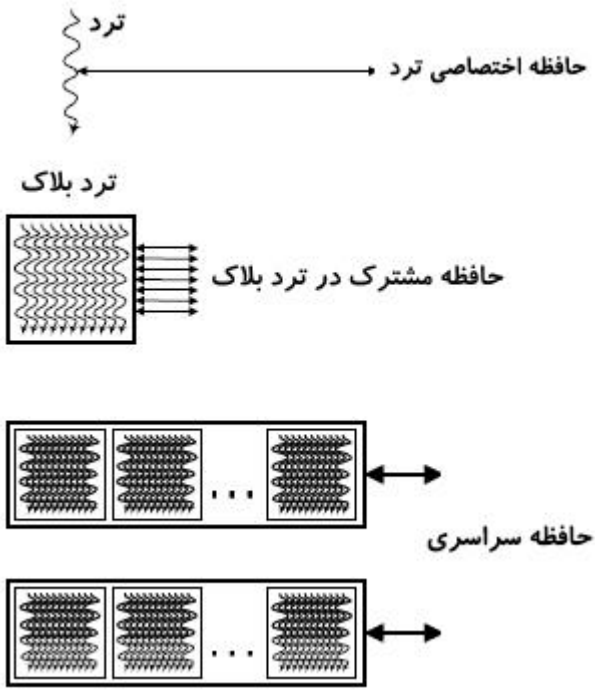
هرچند که پیش از کودا نیز امکان برنامه‌نویسی موازی برای مقاصد عمومی بر روی پردازنده‌های گرافیکی وجود داشت، اما مدلی که کودا ارائه می‌کند، نسبت به مدل‌های پیش‌تر دارای چند ویژگی چشم‌گیر است:

- توانایی دستیابی به آدرس‌های مختلف حافظه برحسب نیاز برنامه‌نویس بدون وجود محدودیت جدی سخت‌افزاری؛ توانایی دستیابی اتفاقی به حافظه.
- در نظر گرفته شدن حافظه‌ی مشترک در بلاک‌ها. این حافظه‌ی مشترک امکان تبادل داده بین بلاک‌ها را با پهنای باند بیشتر و تأخیر کمتری فراهم می‌کند.
- پشتیبانی کامل از عملیات ریاضی بر روی اعداد صحیح[1] و پشتیبانی از عملیات منطقی بیتی.
- هم‌زمان‌سازی منطقی و آسان تردها
- امکان فراگیری آسان

یک برنامه‌ی CUDA متشکل از یک سری کرنل[2] است که ممکن است موازی باهم یا به‌صورت متوالی اجرا شوند. هر کرنل به‌صورت موازی در قالب چندین ترد اجرا می‌گردد. برای راحت‌تر شدن برنامه‌نویسی این تردها به صورت یک سری د سته که به آن‌ها بلاک[3] گفته می شود تق سیم‌بندی می شوند. خود بلاک‌ها هم در قالب یک سری گرید[4] مرتب می شوند. پس هر گرید شامل تعدادی بلاک و هر بلاک شامل تعدادی ترد ا ست. پردازنده‌ی گرافیکی برنامه‌ی کرنل را بر روی تعدادی ترد به‌صورت هم‌زمان اجرا می‌کند. هر ترد به‌تنهایی کل برنامه‌ی کرنل را در اختیار دارد اما با تدابیری که بعداً تو ضیح داده خواهد شد، هر ترد فقط بخ شی از برنامه‌ی کرنل را اجرا می‌کند بدین ترتیب زمانی که کار کل تردها به پایان برسد عملاً اجرای کرنل به پایان رسیده است. هر ترد دارای یک شماره‌ی اختصاصی، یک شمارنده‌ی برنامه[5]، تعدادی رجیستر و حافظه‌ی اختصاصی انحصاری است. در شکل 1-0 این تقسیم‌بندی‌ها نشان داده‌شده است.

[1] نمونه‌های جدید پردازنده‌های گرافیکی منطبق با کودا از اعداد اعشاری نیز به خوبی پشتیبانی می‌کنند.

[2] Kernel
[3] Block
[4] Grid
[5] Program Counter




**شکل 1-0: ترد، ترد بلاک و حافظه‌ی سراسری در پردازنده‌های گرافیکی**

در معماری کودا ساختمان چینش ترد بلاک‌ها در داخل گرید می‌تواند یک‌بعدی یا دوبعدی باشد. درعین‌حال ساختمان تردها هم در داخل ترد بلاک می‌تواند یک، دو یا سه‌بعدی باشد. در فصول آتی در مورد ترد، ترد بلاک، گرید و کرنل بحث خواهیم کرد. در حال حاضر فقط کافی است با این مفاهیم یک آشنایی سطحی داشته باشید برای اینکه بحث به‌روشنی ختم گردد اجازه دهید مفاهیم فوق‌الذکر را مجدداً مرور کنیم.

**کرنل:** به قسمت یا قسمت‌های کوچکی از برنامه که برای اجرا بر روی پردازنده‌ی گرافیکی ارسال می‌گردد کرنل می‌گویند. کرنل قسمتی از برنامه بوده که دارای ویژگی‌های زیر است.

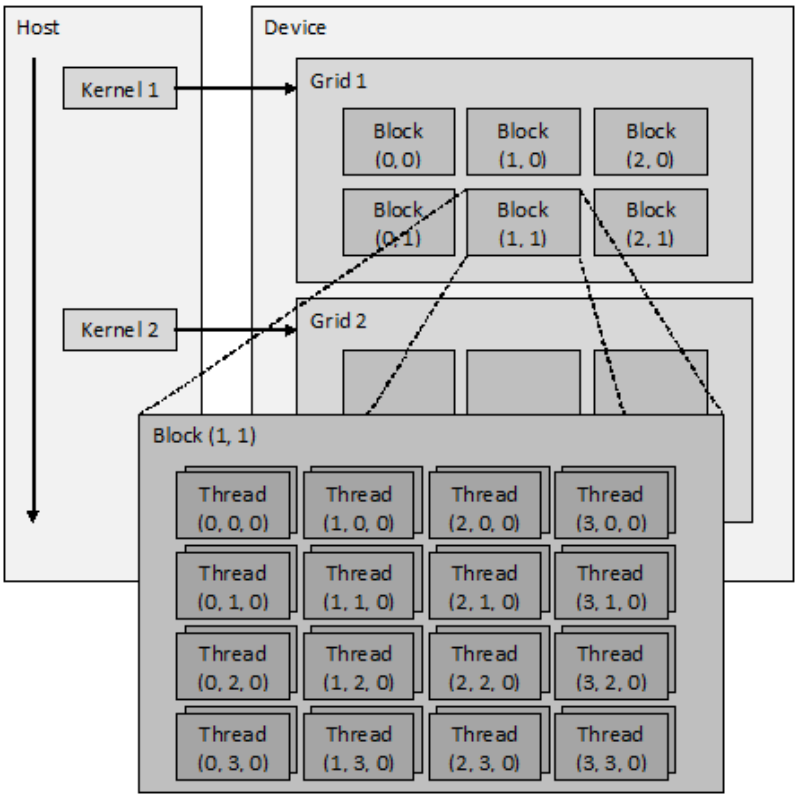
- بتوان به آن به‌صورت یک تابع ایزوله شده نگاه کرد.
- قادر باشد بر روی‌داده‌های مستقل به‌صورت مستقل عمل کند.

**ترد:** به کوچک‌ترین قسمت برنامه که برای اجرای موازی در نظر گرفته شده است ترد می‌گویند. یک Thread از تعدادی دستورالعمل تشکیل می‌شود که به ترتیب و بر روی یک هسته کودا اجرا می‌شود. علت انتخاب نام Thread (به معنای نخ) نیز به همین علت است. درواقع، Thread کوچک‌ترین جزء ساختار سلسله مراتبی مدل

کودا اســت که اجراکننده‌ی دســتورالعمل‌های کرنل اســت. یعنی درنهایت، یک کرنل توســط تعداد زیادی ترد اجرا می‌شود.

**ترد بلاک:** در داخل هر گرید تعدادی ترد بلاک (بلاک) وجود دارد. تردهایی که در داخل ترد بلاک هستند در حین اجرای برنامه می‌توانند باهم داده رد و بدل نمایند.

**گرید:**در داخل هر کرنل فقط یک گرید وجود دارد. گرید یک تعریف اســت که نحوه‌ی اختصــاص منابع به کرنل را نشـــان می‌دهد. به مجموعه‌ای از ترد بلاک‌ها یک گرید می‌گویند. در شـــکل 2-0 رابطه‌ی کرنل و گرید نشـــان داده‌شده است.

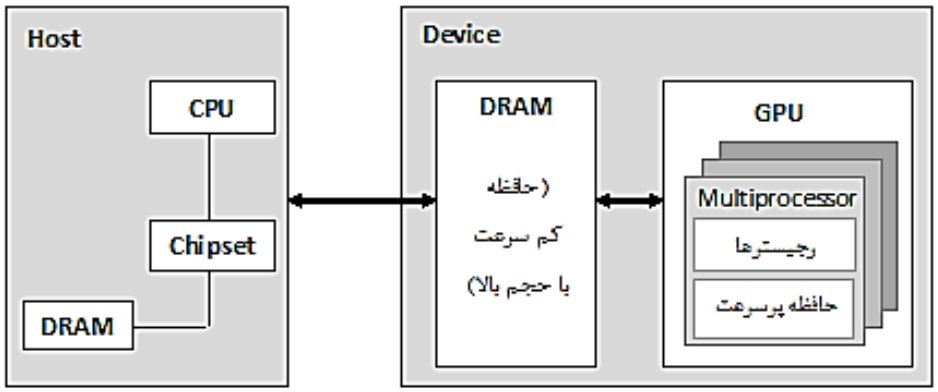



**شکل 2-0: کرنل، گرید، بلاک و ترد در مدل میزبان – وسیله**

پیش‌تر گفتیم که پردازنده‌های قدیمی‌تر دارای دو واحد مهم بودند که عبارت‌اند از واحد پردازش رأســـی و واحد پردازش پیکسلی که به ترتیب مسئول اجرای عملیات سایه زنی روی رئوس و پیکسل‌ها هستند. عملیات سایه زنی شامل تمامی کارهایی است که برای ایجاد سطوح مناسبی از نور بر روی یک تصویر صورت می‌گیرد. عملیات سایه

زنی شامل دو قسمت مهم است. قسمت نخست سایه زنی رئوس[1] است. در این قسمت یکی از وظایف مهم یافتن محل رئوس سه‌بعدی در یک مختصات دوبعدی است. قسمت دیگر که با نام واحد سایه زنی پیکسل[2] شناخته می‌شود نیز برای محاسبه‌ی مقادیر رنگ و سایر مشخصات هر بخش از تصویر به کار گرفته می‌شود. متأسفانه این معماری برای پردازش موازی خیلی مناسب نبود به همین دلیل شرکت NVIDIA سعی کرد معماری پردازنده‌های گرافیکی را به نحوی بهینه کند تا هم امکان اجرای برنامه‌های گرافیکی را به همان صورت قبل داشته باشند هم اینکه بتوان از آن‌ها برای پردازش موازی استفاده شود. اولین تلاش شرکت NVIDIA در این راه منجر به تولید پردازنده‌ی گرافیکیGeForce سری 8 شد.

برای اینکه صحبت در مورد برنامه‌نویسی موازی ناهمگون با مشارکت هم‌زمان CPU و GPU آسان‌تر شود، در بسیاری از مواقع از تشریح روند کار به کمک مدل میزبان[3]، وسیله[4] استفاده می‌شود. این مدل در شکل 3-0 نشان داده شده است. همان‌طور که در شکل نشان داده شده است، پردازنده‌ی اصلی قادر است فقط از طریق حافظه‌ی خارجی به تبادل داده با پردازنده‌ی گرافیکی بپردازد. لازم به ذکر است که هرچند پیاده‌سازی واقعی با این مدل خیلی تفاوت دارد اما این مدل برای به دست آوردن یک درک کلی از روش کار مناسب است. دقت کنید که اولاً سرعت دسترسی به منابع داخلی موجود پردازنده‌ی گرافیکی بسیار بیشتر از سرعت دسترسی به منابع مشترک است. ثانیاً پردازنده‌ی اصلی نمی‌تواند به منابع داخلی پردازنده‌ی گرافیکی دسترسی پیدا کند و بالعکس.




**شکل 3-0: مدل میزبان، وسیله**

[1]Vertex Shading
[2]Pixel Shading
[3] Host
[4] Device

## 2.3 حافظه در کودا

در یک پردازنده‌ی گرافیکی منطبق بر کودا، شـــش نوع حافظه‌ی مختلف با کاربردهای متفاوت وجود دارد. این حافظه‌ها را می‌توان به دودسته‌ی حافظه‌های روی تراشه و حافظه‌های بیرون تراشه تقسیم کرد. حافظه‌های بیرون تراشه ازنظر فیزیکی جزئی از یک DRAM محسوب شده و لذا ویژگی‌های DRAM را دارند یعنی دارای حجم بالا و سرعت پایین (درصورتی‌که cache نشوند، بین 200 الی 300 سیکل) می‌باشند. این حافظه‌ها شامل موارد زیر هستند.

- **حافظه سراسری[1]:** هر پردازنده‌ی گرافیکی یک نمونه از این حافظه در اختیار دارد. این حافظه از سمت پردازنده‌ی گرافیکی هم قابل‌خواندن و هم قابل‌نوشـــتن اســـت. شـــایان‌ذکر اســـت که این حافظه برخلاف حافظه‌های ثوابت و بافت دارای حافظه‌ی نهان نیست.

- **حافظه ثوابت[2]:** هر پردازنده‌ی گرافیکی یک نمونه از این حافظه را در اختیار دارد. این حافظه از ســـمت پردازنده‌ی گرافیکی فقط قابل‌خواندن اســـت. حافظه‌ی ثوابت همان‌طور که از نامش پیداســـت، حافظه‌ای است برای ذخیره کردن مقادیر ثابت. این مقادیر، می‌توانند آرایه‌ای از اعداد باشند که قبلاً توسط پردازنده‌ی اصلی به این حافظه منتقل شده و ترد آن‌ها را می‌خوانند. طول عمر اطلاعات موجود در حافظه ثوابت برابر با طول عمر برنامه است و در طول مدتی که کرنل‌های مختلفی از یک برنامه اجرا می‌شوند، حافظه ثوابت مقادیر خود را حفظ می‌کند. این حافظه، دارای حافظه‌ی نهان نیز هســـت. حافظه‌ی نهان برای کاربردهایی که در آن‌ها پردازنده‌ی گرافیکی به‌صـــورت مکرر اعدادی را از حافظه‌ی خود می‌خواند، اســـتفاده می‌گردد. سرعت دسترسی به اطلاعات موجود در حافظه نهان برابر با سرعت دسترسی به حافظه مشترک است پس به صورت کلی سرعت این حافظه می‌تواند بین 1 تا 300 سیکل ساعت تغییر کند. باید توجه داشت که استفاده‌ی اشتباه و یا بیش‌ازحد نیاز از این حافظه تنها احتمال عدم اصابت[3] را زیاد می‌کند.

- **حافظه بافت[4]:** هر پردازنده‌ی گرافیکی دارای یک حافظه‌ی بافت با دسترسی خواندن و نوشتن است. کلمه‌ی بافت در اصطلاح گرافیکی به معنای تصویری است که بر روی سطح اشیای سه‌بعدی نمایش داده می‌شـــود. در کودا این حافظه نیز کاربردهای منحصـــر به خود را دارد. این حافظه نیز دارای حافظه‌ی نهان است. با توجه به اینکه این حافظه، جزئی از DRAM به‌حساب می‌آید، سرعت دسترسی به اطلاعاتی از آن‌که در حافظه‌ی نهان موجود نیست بسیار پایین است. این حافظه برای حالتی بهینه‌سازی شده است که

[1] Global memory
[2] Constant memory
[3] Cache miss
[4] Texture memory

بخواهیم آرایه‌های دوبعدی ذخیره کنیم و تردهای بخواهند درایه‌هایی نزدیک به هم را از آن بخوانند. برای حافظه بافت، منطق ذخیره‌سازی اطلاعات در حافظه‌ی نهان مطابق با نیازهای پردازش گرافیکی بافت‌ها است. علاوه بر این، حجم حافظه بافت قابل تنظیم است و حداکثر می‌تواند برابر با کل فضای DRAM باشد. اما استفاده‌ی بیش‌ازحد از آن، تنها باعث کاهش نرخ اصابت[1] و سرعت خواندن از آن می‌شود.

- **حافظه محلی[2]:** هر ترد بخشی از این حافظه را، به صورت انحصاری، برای خواندن و نوشتن در اختیار دارد. هرگاه اطلاعات یک ترد در رجیسترها جا نشود و نخواهیم این اطلاعات را در حافظه‌ی مشترک قرار دهیم باید این اطلاعات را به حافظه‌ی محلی سپرد. حافظه‌های محلی بر روی DRAM قرار دارند پس دارای حجم بسیار بزرگ‌تری نسبت به حافظه‌ی مشترک می‌باشند. هر ترد به بخشی از حافظه‌ی DRAM تحت عنوان حافظه‌ی محلی دسترسی دارد. شایان‌ذکر است که پس از پایان اجرای کرنل و از بین رفتن ترد، قسمتی از حافظه که در اختیار این ترد بود نیز از بین خواهد رفت یعنی اطلاعات آن دیگر قابل‌دسترسی نخواهد بود.

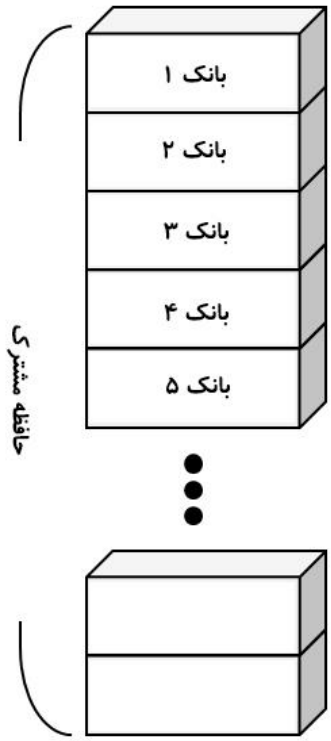
دسته‌ی دوم حافظه‌های روی تراشه هستند. این حافظه‌ها دارای سرعتی بالاتر (یک سیکل) و حجمی پایین‌تر (حافظه‌ی مشترک[3] معمولاً 16 یا 48 کیلوبایت بوده و ثبات‌ها معمولاً 16384 یا 8192 عدد هستند[4]) هستند.

- **حافظه مشترک:** همه‌ی تردهای موجود در یک بلاک، به‌صورت اشتراکی از یک نمونه از این حافظه برای خواندن و نوشتن استفاده می‌کند.
- **ثبات‌ها:** هر ترد می‌تواند از تعدادی ثبات داخلی برای خواندن و نوشتن استفاده کند.

چون تردهای موجود در یک بلاک باهم از حافظه‌ی مشترک استفاده می‌کنند، این حافظه نیاز به پهنای باند زیادی دارد که بتواند به همه‌ی این تقاضاها پاسخ دهد. برای دسترسی به این پهنای باند حافظه‌های مشترک به تعدادی بانک تقسیم‌شده‌اند. این بانک‌ها دارای ورودی و خروجی مستقل و طول برابری هستند. برای مثال در پردازنده گرافیکی G80، هر حافظه مشترک 16 کیلوبایت بوده و از 16 بانک یک کیلوبایتی با کلمات 32 بیتی تشکیل می شود. هر بانک می‌تواند در هر سیکل به یک آدرس سرویس‌دهی کند. به‌عبارتی‌دیگر پهنای باند هر بانک، 32 بیت در هر سیکل ساعت است. بنابراین تعداد سرویس‌های هم‌زمانی که حافظه مشترک می‌تواند بدهد، به تعداد بانک‌هایش است.

[1] Cache hit
[2] Local memory
[3] Shared Memory
[4] واضح است که این اعداد در آینده تغییر خواهند کرد.

شـکل 4-0 نحوه‌ی تقسـیم یک حافظه‌ی مشـترک را به بانک‌های ذکرشـده، نمایش می‌دهد. کلمات 32 بیتی متوالی، به بانک‌های متوالی تخصیص داده می‌شوند.




**شکل 4-0: بانک‌های حافظه در کودا**

اگر درآن‌واحد، چندین ترد تقاضای دسترسی به یک بانک را داشته باشند، تداخل پیش می‌آید و پردازنده‌ی گرافیکی ناچار است که به تقاضاها، به‌صورت نوبتی پاسخ دهد. در صورت بروز تداخل بانک، سرعت نهایی حافظه مشترک افت می‌کند. تقسـیم حافظه مشـترک به بانک‌های متعدد، احتمال تداخل در دسـترسـی به بانک‌ها را کاهش داده و سرعت نهایی خواندن و نوشتن در آن را افزایش می‌دهد.

در حالت عادی، تداخلی بین Threadهای دونیمه‌ی Warp وجود ندارد؛ زیرا Thread های نیمه‌ی دوم Warp صبر می‌کنند تا Thread های نیمه‌ی اول کار شان با Shared Memory تمام شود. هرچند این اتفاق می‌تواند نوعی تداخل بانک محسوب شود، اما نظر به اینکه هر Warp شامل 32 Thread بوده و تنها 16 بانک مستقل برای پاسـخ‌دهی به نیازهای این Thread ها وجود دارد، بنابراین تداخل بین دونیمه‌ی یک Warp امری اسـت اجتناب‌ناپذیر و معمولاً از تأخیر به وجود آمده صرف‌نظر می‌شود.

با توجه به این‌که آدرس‌های یک Shared Memory معمولاً 32 بیتی هستند، در حالتی که با آرایه‌ای سروکار داریم که اعضای آن‌ها متغیرهای غیر 32 بیتی (مثل char یا int64) هستند، بایستی مراقب تداخل در بانک‌ها باشیم.

دو شکل 4-7، حالتی را معرفی می‌کنند که هیچ تداخلی در دسترسی Thread ها به بانک‌های Shared Memory وجود ندارد. شکل سمت چپ آدرس‌دهی خطی با گام یک، و شکل سمت راست، آدرس‌دهی تصادفی یک‌به‌یک را نشان می‌دهد.

## 2.4 معماری پردازنده های گرافیکی GeFroce سری 8

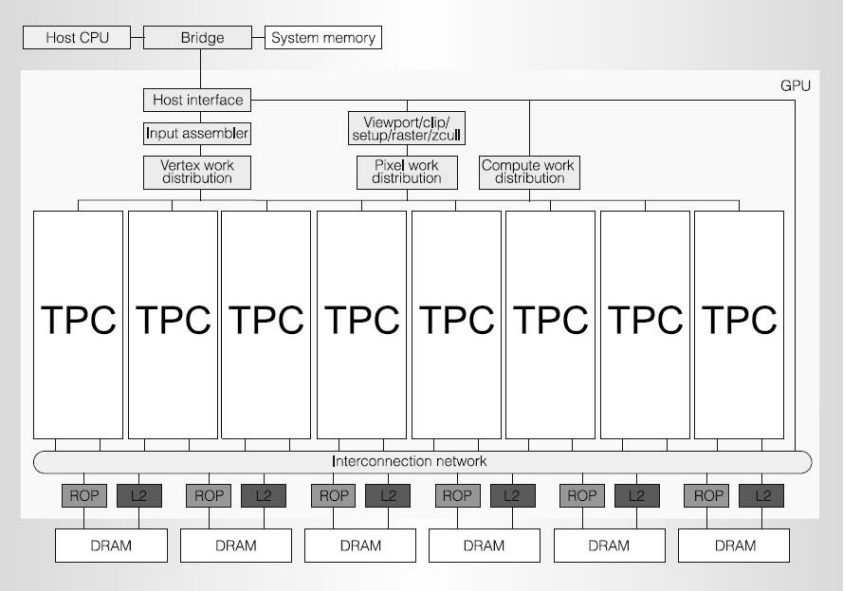
پردازنده‌های گرافیکی سری 8800 که موسوم به G80 هستند در آبان ماه سال 1385 همراه با ارائه کارت‌های گرافیکی GeForce 8800 GTX مطابق با معماری کودا ساخته و به بازار ارائه شدند. یکی از مهم‌ترین تغییراتی که در معماری G80 اتفاق افتاده بود این بود که در این پردازنده‌ی گرافیکی برای نخستین بار واحد پردازش‌های رأسی و پیکسلی با یکدیگر ادغام شدند و یک واحد یکتا به وجود آمد که قادر بود همه‌ی پردازش‌های رأسی[1]، پیکسلی[2]، هندسی و پردازش‌های موازی را باهم انجام دهد. دلایلی که شرکت NVIDIA را بدین سمت پیش برد در ادامه توضیح داده‌شده است.

- **توزیع بار ناهمگون:** یکی از مهم‌ترین علل ادغام این دو واحد وجود توزیع بار ناهمگون است. در پردازش تصاویر، زمانی که با زوایای بزرگ روبه‌رو هستیم، واحد پردازنده‌ی رئوس در بسیاری از مواقع بیکار است و بار زیادی بر دوش واحد پردازش پیکسلی قرار دارد. در عوض در زمان‌هایی که مشغول پردازش زاویه‌های کوچک هستیم، بار زیادی بر دوش واحد پردازش رأس قرار دارد و واحد پردازش پیکسلی بیکار است. درصورتی‌که برای هرکدام از این واحدها یک سخت‌افزار مجزا تعریف کنیم، نمی‌توانیم هر دو واحد را زیر بار کامل ببریم پس طراحی هرگز بهینه نخواهد بود.
- **کاهش هزینه:** دو واحد تلفیق‌شده می‌توانند از بسیاری از بخش‌ها به‌صورت مشترک استفاده کنند. این استفاده مشترک مساحت تراشه و هزینه‌های طراحی را کاهش می‌دهد.
- **طراحی یکپارچه:** با تلفیق این دو واحد یک گروه طراحی می‌تواند به‌راحتی بر روی جنبه‌های مختلف طراحی تمرکز نماید و طرحی بهتر و یکپارچه‌تر ارائه دهد.

[1] Vertex process
[2] Pixel process

- **عمومیت در برنامه‌نویسی:** واحد پردازنده‌ی جدیدی که به‌جای دو پردازنده‌ی قبلی قرارگرفته است می‌تواند برنامه‌های عمومی‌تری را اجرا کند نه‌فقط برنامه‌هایی که برای پردازش تصاویر نوشته‌شده است. بدین‌سان پردازش موازی بر روی پردازنده‌ی گرافیکی آسان خواهد شد.

در شکل 5-0 معماری یک پردازنده‌ی گرافیکی سری 8 نشان داده‌شده است. روند پردازش اطلاعات در این پردازنده از بالا به پایین است؛ اطلاعات خام از قسمت بالای پردازنده وارد شده، پردازش می شود و نهایتاً از قسمت پایین شکل، اطلاعات پردازش‌شده خارج می‌شود. در ادامه واحدهای مختلف این پردازنده شرح داده‌شده است.




شکل 5-0: معماری پردازنده‌ی گرافیکی GeForce 8800

1- آرایه پردازنده‌های جریان (SPA[1])

واحد SPA که در شکل نشان داده نشده است، شامل مجموعه‌ی تمام واحدهای TPC[2] است.این واحد مسئولیت اجرای برنامه‌های گرافیکی، اجرای برنامه‌های پردازش موازی، کنترل و مدیریت تردها را بر عهده دارد.

2- رابط میزبان[3]

در ادبیات پردازش موازی با پردازنده‌های گرافیکی، به پردازنده‌ی اصلی میزبان و به پردازنده‌ی گرافیکی وسیله[4] گفته می شود. بنابراین، واحد رابط میزبان همان‌طور که از اسمش پیداست وظیفه‌ی هماهنگی با میزبان را بر عهده

[1]Streaming Processor Array
[2]Texture/Processor Cluster; Thread Processing Cluster
[3]Host Interface
[4]Device

دارد. این واحد باید دستورات CPU را دریافت کرده، اطلاعات موردنیاز را از حافظه فرابخواند و سازگار بودن دستورات مختلف را بررسی کند.

3- واحد Input Assembler

وظیفه‌ی این واحد واکشی اطلاعات مربوط به تصاویر از حافظه است. این واحد قادر است در هر پالس ساعت یک مشخصه‌ی هندسی و هشت مشخصه‌ی رأسی را از حافظه برای پردازش واکشی کند.

4- واحد Viewport/Clip/Setup/Raster/Zcull

این واحد وظیفه دارد تا یک سری پیش‌پردازش مقدماتی بر روی تصاویر انجام دهد. مهم‌ترین مسئولیت این قسمت تبدیل کردن اطلاعات برداری تصاویر به اطلاعات پیکسلی برای پردازش تصویر است.

5- واحدهای توزیع کار

واحدهای توزیع کار رأسی[1]، پیکسلی[2] و پردازشی[3] مسئولیت توزیع پردازش‌های مختلف را در بین واحدهای TPC بر عهده دارند. وظیفه‌ی این واحدها توزیع بار یکنواخت بین TPCهاست.

6- واحد ROC[4]

هر واحد ROC به یک قسمت مشخص از حافظه دسترسی دارد. TPC اطلاعات را از طریق شبکه‌ی اتصالات داخلی[5] برای ROC ارسال می‌کند.وظیفه‌ی ROC کنترل عمق رنگ، تست رنگ، مخلوط کردن و به‌روزرسانی رنگ‌هاست. واضح است که تمامی این واحدها به‌صورت موازی باهم کار می‌کنند.

7- DRAM

در پردازنده‌های GeForce سری 8 حافظه‌ی RAM به‌صورت 8 بانک مستقل در نظر گرفته‌شده است. این سبک طراحی RAM کمک می‌کند تا پهنای باند بیشتری در اختیار پردازنده‌ها قرارگرفته و عملیات ذخیره و بازیابی به‌صورت موازی انجام شود. در معماری سری 8800 انتقال اطلاعات از RAM به واحد SPA از طریق کش L2 صورت می‌گیرد. تعداد پایه‌های باس داده در RAM برابر با 384 پایه است که به شش گروه که هرکدام دارای 64 پایه‌ی داده هستند تقسیم می‌گردد. شایان‌ذکر است که حافظه‌ی محلی[6] و حافظه‌ی سراسری[7] بر روی این

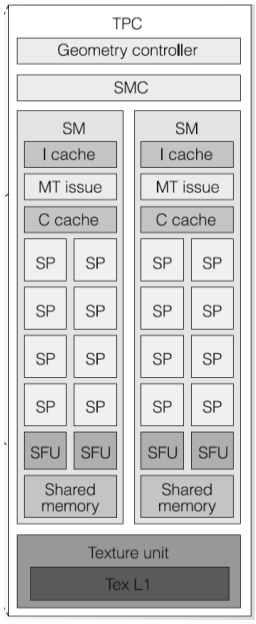
[1]Vertex Work Distribution
[2]Pixel Work Distribution
[3]Compute Work Distribution
[4]Raster Operation Processor
[5]Interconnection Network
[6]Local Memory
[7]Global Memory

RAM قرار دارند. حافظه‌ی محلی یک حافظه‌ی انحصاری است که برای هر ترد به‌صورت مستقل قابل‌دسترسی است. به‌عبارت‌دیگر تردهای مختلف نمی‌توانند به حافظه‌ی محلی همدیگر دسترسی پیدا کنند. حافظه‌ی عمومی هم حافظه‌ای است که برای همه‌ی تردها قابل‌دسترسی است. تردهای مختلف می‌توانند اطلاعاتشان را در حافظه‌ی عمومی باهم تبادل کنند.

### 8- واحد TPC

همان‌طور که در شکل 6-0 نشان داده‌شده است، هر واحد TPC دارای یک کنترل‌کننده‌ی اطلاعات هندسی[1]، یک واحد کنترل‌کننده‌ی SM[2]، یک واحد بافت[3] و دو SM[4] است.واضح است که تعداد المان‌هایی که در داخل هر SM قرار دارد می‌تواند برحسب نیاز تغییر کند. واحد کنترل‌کننده‌ی اطلاعات هندسی موظف است خط لوله‌ی منطقی را که برنامه‌ی پردازش تصویر بر اساس آن نوشته شده است بر روی سخت‌افزار SMها بنگارد. واحد SMC موظف است که SMها مختلف را مدیریت کرده و بین آن‌ها برای دسترسی به منابع مشترک از قبیل واحد بافت، دسترسی به حافظه و ورودی‌ها و خروجی‌ها داوری نماید. واحد SMC مسئول توزیع عادلانه بار محاسباتی بین SMها نیز است. در یک TPC چند SM نیز وجود دارد که این SMها وظیفه‌ی اجرای پردازش‌های رأسی، پیکسلی، هندسی و پردازش‌های موازی عمومی را بر عهده‌دارند. آخرین واحدی که در یک TPC قرار دارد واحد بافت است. واحد بافت قادر است در هر سیکل یک دستورالعمل را اجرا نماید. ورودی واحد بافت مختصات پیکسل‌هایی است که قرار است بر روی آن‌ها عمل پردازش صورت گیرد و خروجی آن اطلاعات فیلتر شده است. واحد بافت به‌صورت کاملاً مستقل و موازی نسبت به SM عمل می‌کند. هر واحد پردازش بافت دارای چهار واحد تولیدکننده‌ی آدرس و هشت واحد فیلترکننده است که همه‌ی آن‌ها قادرند به‌صورت موازی کار کنند. واحد بافت کاملاً به‌صورت خط لوله‌ای پیاده شده است و دارای یک حافظه‌ی پنهان است که پیاده کردن فیلترهای محلی را تسریع می‌کند.

[1]Geometry Controller
[2]SM Controller (SMC)
[3]Texture Unit
[4]Streaming Multiprocessor




شکل 6-0: TPC

مهم‌ترین بخش TPC واحد SM ا ست. این واحد دارای یک کش د ستورالعمل[1]، یک کش فقط خواندنی ثوابت[2]، یک واحد واکشـی و اجرای دسـتورالعمل[3]، 16 کیلوبایت حافظه‌ی مشـترک خواندنی و نوشـتنی، هشـت پردازنده جریان[4]، و دو واحد پردازش‌های ویژه[5] ا ست.هرکدام از SPها دارای یک واحد MAD[6] ا ست که عملیات ضرب و جمع متوالی به فرم $A = B \times C + D$ را در یک پالس سـاعت انجام خواهد داد.SPها یا پردازنده‌های جریان به‌تدریج با نام هسـته‌های کودا شـناخته شـدند. پس‌ازاین ما نیز برای خطاب کردن پردازنده‌های جریان (SP) لفظ هسته‌های کودا را به کار خواهیم برد.

در هر SM واحد SFU دو وظیفه‌ی عمده بر عهده دارد. وظیفه‌ی نخ ست اجرای عملیات ریا ضی سطح بالا مانند عملیات محاسبه‌ی توابع مثلثاتی و جذر بوده و وظیفه‌ی دوم الحاق پیکسل‌ها برای نمایش است. آخرین قسمت یک SM حافظه‌ی م شترک ا ست. حافظه‌ی م شترک به برنامه‌هایی که بر روی ه سته‌های کودای موجود در یک SM اجرا می‌شـوند این امکان را می‌دهد که بدون نیاز به دسـترسـی به حافظه‌ی خارجی به تبادل اطلاعات بپردازند. بدین سان ترافیک د ستر سی به حافظه‌ی خارجی کاهش‌یافته و سرعت نهایی اجرای برنامه افزایش خواهد یافت.در GeForce 8800 فرکانس پالس سـاعت هسـته‌های کودا و واحدهای SFU برابر با 1.5 GHz اسـت و قدرت محا سباتی هر SM در بی شترین حالت برابر با 36 GFLOPS ا ست. می‌توان گفت کوچک‌ترین واحد پردازنده‌ای

[1]Instruction Cache (I cache)
[2]Constant Cache (C cache)
[3]Multithread Instruction Fetch And Issue (MT issue)
[4]Streaming Processor
[5]Special Function Unit
[6]Multiply ADD

که در یک پردازنده‌ی گرافیکی قرار دارد هسته‌های کودا است. تردهایی که برای اجرا به یک SM داده می‌شود بین هسته‌های کودا تقسیم‌شده و هر هسته‌ی کودا یک سری از آن‌ها را پردازش می‌کند به همین دلیل به این هسته‌های کودا پردازنده‌ی ترد هم گفته می شود. یک هسته‌ی کودا قادر است عملیات ریاضی ممیز شناور شامل جمع، ضرب و MAD انجام دهد. این عملیات با استاندارد IEEE 754برای اعداد اعشاری ممیز شناور با دقت ساده منطبق بوده و قادر است مقادیر خاص ازجمله NaN[1] یا مقادیر بی‌نهایت را تشخیص دهد. این هسته همچنین قادر است عملیات ریاضی صحیح متنوعی شامل مقایسه و تبدیل مقادیر انجام دهد. شایان‌ذکر است که این هسته‌ها به‌صورت کاملاً خط لوله‌ای طراحی‌شده‌اند.

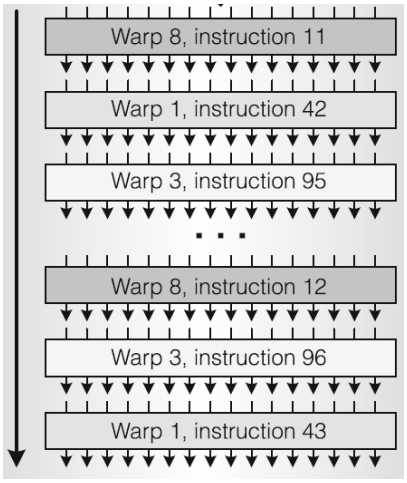
یکی از معایب پردازنده‌های سری 8800 این است که در آن‌ها از اعداد زیر نرمال[2] پشتیبانی نمی‌شود. به‌عبارت‌دیگر در این پردازنده‌ها اعداد زیر نرمال با مقدار صفر تقریب زده‌شده و دور ریخته می‌شوند که این عمل دقت نهایی محاسبات را پایین خواهد آورد.

در کودا، کرنل برنامه‌ی ساده‌ای است که مشخص می‌کند یک ترد چگونه نتایج موردنظر را محاسبه کند. برای اینکه پاسخ یک مسئله داده شود باید تردهای بسیار زیادی به‌صورت هم‌زمان اجرا شوند. حال مسئله‌ای که ایجاد می‌شود این است که چگونه می‌توان اجرای موازی تردهای بسیار زیادی را که ممکن است مربوط به برنامه‌های مختلفی باشد مدیریت کرد. ازآنجایی‌که هر برنامه‌ای که قرار باشد این تردها را مدیریت کند خودش زمان‌گیر است شرکت NVIDIA برنامه‌ی مدیریت اجرای این تردها را به‌صورت سخت‌افزاری پیاده کرده است این برنامه قادر است اجرای 768 ترد هم‌زمان را با سربار[3] زمانی صفر ثانیه مدیریت کند. برای اینکه از برنامه‌های مستقل پردازش رئوس، گرافیک و برنامه‌های پردازش موازی پشتیبانی شود، هر SM قادر است مستقل از سایر SMها برنامه‌ی مشخصی را اجرا کند مشکلی که ممکن است ایجاد شود این است که اجرای قسمت‌های مختلف برنامه بر عهده‌ی SMهای مختلفی گذاشته شود و جایی که نیاز به تبادل داده است کار برخی از SMها به پایان نرسیده باشد. برای حل چنین مشکلی دستورالعملی برای هم‌زمان کردن SM در مواقع نیاز در نظر گرفته‌شده است.در پردازنده‌های گرافیکی NVIDIA از یک معماری جدید به نام SIMT[4] استفاده می‌شود. واحد مدیریت اجرای تردهای SM، تردها را در دسته‌های 32تایی که به آن وارپ[5] گفته می‌شود اجرا می‌کند. واژه‌ی وارپ هم مثل ترد از صنعت نساجی به عاریه گرفته شده است و به معنی تار پارچه است. تار پارچه مجموعه‌ای از یک سری نخ موازی با هم است و اگر به شکل 7-0 توجه کنید متوجه خواهید شد که علت این نام‌گذاری چیست.

[1] Not A Number
[2] Sub normal
[3] Overhead
[4] Single Instruction Multiple Thread
[5] Warp

شکل 7-0: نمونه‌ی تار پارچه

هر SM قادر ا ست 24 وارپ را به صورت هم‌زمان اجرا کند. تمامی تردهایی که در یک وارپ قرار دارند م شابه هم هســـتند و باید از یک آدرس برنامه شـــروع به اجرا کنند اما آن‌ها آزادند که در طول اجرای برنامه به قســـمت‌های مختلفی جهش کرده و به صورت م ستقل اجرا شوند؛ هرچند که اگر چنین اتفاقی رخ دهد اجرای آن‌ها به صورت موازی نخواهد بود. به این ســـبک اجرای موازی تردها SIMT گفته می‌شـــود. در هر SM واحد اجرای SIMT بررســی می‌کند و یکی از 24 وارپی را که برای اجرا آماده اســت انتخاب می‌کند.اولین انتخاب بر اســاس نوع وارپ، نوع د ستورالعمل و رعایت منصفانه بودن شانس اجرا برای همه‌ی وارپ‌ها صورت می‌گیرد. در معماری GeForce سری 8 هر وارپ به‌صورت دو سری 16 تردی اجرا می‌گردد که درمجموع 4 سیکل طول می‌کشد. نمونه‌ی اجرای SIMT در شکل 8-0 نشان داده‌شده است.




شکل 8-0: اجرای SIMT

پس‌ازاینکه یک وارپ برای اجرا انتخاب شد SM، تردهای آن را بر روی ه سته‌های کودا به صورت م ستقل از هم اجرا می‌کند. بهترین بازده، زمانی به دست می‌آید که برنامه‌ای که قرار است اجرا گردد دارای انشعاب وابسته به داده نبا شد. در صورتی‌که چنین پر شی وجود دا شته با شد ا صطلاحاً می‌گویند واگرایی اتفاق افتاده ا ست. هر زمان که در

یک SM واگرایی اتفاق بیفتد تردهای آن به‌صـــورت متوالی اجرا خواهند شـــد. اگر بخواهیم دو روش SIMT و SIMD را باهم مقایســـه کنیم باید گفت که روش اول کنترل ترد را بر عهده دارد ولی روش دوم کنترل برداری از داده‌ها را بر عهده می‌گیرد. تفاوت مهمی که وجود دارد این است که SIMT نه‌تنها اجازه‌ی نگارش برنامه‌ی موازی در سطح داده را می‌دهد بلکه اجازه‌ی اجرای تردهای مستقل را نیز فراهم می‌کند.

کامپایلرهای ســـطح بالا برنامه‌های گرافیکی و برنامه‌های پردازش موازی را مســـتقیماً به برنامه‌ی باینری قابل‌اجرا توسط پردازنده‌ی گرافیکی تبدیل نمی‌کنند بلکه آن را به یک سری دستورالعمل میانی مانند دستورالعمل‌های PTX تبدیل کرده و سپس این دستورالعمل‌ها پس از بهینه سازی به دستورالعمل‌های قابل‌اجرا توسط پردازنده‌ی گرافیکی تبدیل می‌شـــوند. بدین ترتیب شـــرکت می‌تواند نســـل‌های متفاوتی از پردازنده‌های گوناگون تولید کند ولی زبان برنامه‌نویســی را تغییر ندهد. لازم به ذکر اســت که تغییر مداوم زبان برنامه‌نویســی، کاربران را نســبت به اســتفاده از سخت‌افزار دلسرد می‌کند زیرا کاربر هر روز مجبور است زبان جدیدی را بیاموزد. فایده‌ی دیگر این سبک کامپایل برنامه این است که در آن می‌توان برنامه را قبل از تبدیل نهایی به کدهای باینری بررسی کرد، وابستگی‌های داده‌ها را بررسی کرد، محل‌های وقوع واگرایی را تشخیص داد، کدهای مرده (قسمت‌هایی که هرگز اجرا نخواهند شد) را کشف کرد و برنامه را تا حد امکان برای اجرا بر روی پردازنده‌ی گرافیکی بهینه کرد.

هر پردازنده‌ای متناسب با کاربردی که دارد دارای یک مجموعه دستورالعمل خاص است که در راستای هدف آن پردازنده بهینه شده‌اند. در ادامه مجموعه‌ی دستورالعمل‌هایی که توسط پردازنده‌های گرافیکی سری 8800 شرکت NVIDIA پشتیبانی می‌شوند آورده شده است.

- مجموعه دستورالعمل‌های اعشاری و صحیح

  - این دســتورها شــامل جمع، ضــرب، ضــرب–جمع یا همان MAD، یافتن کمینه، بیشــینه، مقایســه و دســتورهای تبدیل بین اعداد صــحیح و اعشــاری هســتند. همچنین یک ســری دستورات دیگر هم هست که توسط واحد SFU اجرا می شود. این دستورات عبارت‌اند از محاســـبه‌ی ســـینوس، کســـینوس، مقدار نمایی، لگاریتم باینری، معکوس کردن عدد و محاسبه‌ی جذر.

  - این عملیات قادرند پرچم‌های[1] صفر، منفی، کری[2] و سرریز[3] را که به صورت اختصاصی در هر ترد قرار دارند برای استفاده‌های بعدی ست کنند.

[1] Flag
[2] Carry
[3] Overflow

- عملیات بیتی
  - عملیات بیتی شامل شیفت به سمت راست، شیفت به سمت چپ، عملیات منطقی و جابه‌جایی هستند.
- عملیات کنترل اجرا
  - این عملیات شامل انشعاب، فراخوانی، بازگشت، دستورالعمل trap و دستور مربوط به سنکرون کردن تردها است.
- دستورالعمل‌های دسترسی به حافظه
  - برای دسترسی به حافظه دو نوع دستورالعمل در نظر گرفته‌شده است. نوع نخست دستورالعمل‌های بافت است که با کمک خود واحد بافت می‌تواند به اطلاعات حافظه دسترسی پیدا کند. نوع دیگر دستورالعمل‌های ذخیره و بازیابی معمولی است. این دستورالعمل‌ها برای هماهنگ شدن کامل سخت‌افزار با زبان C به آن اضافه‌شده است. این دستورها توسط SM اجرا می‌گردند.

تا اینجا با معماری پردازنده‌های گرافیکی سری 8 و معماری کودا آشنا شدیم. اکنون زمان مناسبی است که بین این دو معماری ارتباط برقرار کنیم و ببنیم که مدلی که کودا برای برنامه‌نویسی ارائه می‌کند چگونه بر روی سخت‌افزار پردازنده‌های گرافیکی نگاشته می‌شود.

گفته شد که برای اجرای یک برنامه‌ی موازی نیاز است تعداد بسیار زیادی ترد اجرا شوند و هرکدام از آن‌ها کار خاصی را انجام دهند تا نهایتاً پاسخ مسئله به دست آید. همچنین گفته شد که این تردها ممکن است در SMهای گوناگون باشند و هر SM این تردها را در قالب‌هایی به نام وارپ اجرا می‌کند. فرض کنید فقط یک برنامه بر روی پردازنده‌ی گرافیکی در حال اجرا است؛ برای اجرای این برنامه یک سری ترد در حال اجراشدن هستند به این تردها که همه مربوط به اجرای یک برنامه بوده و برای یافتن پاسخ ممکن است به مبادله‌ی اطلاعات نیاز داشته باشند. CTA[1] گفته می‌شود. تعداد تردهای درون یک CTA می‌تواند از 1 تا 512 عدد باشد. هرکدام از این تردها دارای یک شماره‌ی منحصربه‌فرد در درون CTA هستند و برنامه‌نویس می‌تواند تردها را درون CTA به‌صورت یک‌بعدی، دوبعدی و یا سه‌بعدی بچیند. تردهایی که درون یک CTA هستند قادرند داده‌هایشان را از طریق حافظه‌ی سراسری و یا محلی باهم مبادله کنند و یا در نقطه‌های خاصی از اجرا باهم سنکرون گردند. هر SM قادر

[1]Cooperative Thread Array

است نهایتاً تا هشت CTA را به‌صورت هم‌زمان اجرا کند. این تعداد وابسته است به منابعی که هر CTA نیاز دارد. واحد SMC درصـــورتی‌که ببیند منابع خالی وجود دارد یک CTA تعریف کرده و تردهای جدیدی را اجرا می‌کند. واضح است که SM، هر CTA را در قالب وارپ‌هایی که هرکدام دارای 32 ترد است اجرا خواهد کرد. این طراحی خارق‌العاده به اینجا ختم نمی‌شود؛ در پردازنده‌های سری 8 مجموعه‌ای از CTAها قرار دارد که به آن گرید CTA گفته می‌شـــود. در این گرید CTAهای زیادی قرار دارد که واحد توزیع کار این CTAها را با توجه به اینکه کدام SM دارای ظرفیت خالی است بین آن‌ها توزیع می‌کند. برای اینکه بتوان فارغ از اندازه پردازنده‌ی گرافیکی به‌راحتی برنامه‌نویســی کرد، CTAها کاملاً مســتقل از هم اجرا می‌شــوند و تردهایی که در داخل CTAهای مختلف قرار دارند امکان تبادل اطلاعات را با هم نخواهند داشت. بدین‌سان ممکن است تمام CTAهای یک برنامه‌ی خاص بر روی یک پردازنده‌ی گرافیکی به‌صـورت موازی باهم اجرا شـوند و بر روی پردازنده‌ی گرافیکی دیگر هم به‌صـورت متوالی اجرا شوند در این حالت تنها تفاوتی که حا صل می شود در زمان اجرای برنامه ا ست و ا شتباهی در خروجی پدید نخواهد آمد. پس سلسله‌مراتب اجرا در پردازنده‌ی گرافیکی دارای سه قسمت زیر است:

- ترد: یک بخش از پاسخ را بیان می‌کند و با TID مشخص می‌گردد.
- CTA: پاسخ بلاک‌هایی از برنامه را مشخص می‌کند و با CTA ID مشخص می‌شود.
- Grid: پاسخ تعداد زیادی CTA را مشخص می‌کند.

با این تفاســیر تطبیق مدل برنامه‌نویســی کودا با معماری ســخت‌افزاری کار دشــواری نیســت. هر ترد در مدل برنامه‌نویسی کودا با یک ترد در سخت‌افزار معادل است. هر بلاک کودا معادل CTA و گرید کودا با گریدی که در این قسمت معرفی شد برابر است. پس بلاک‌هایی که در کودا تعریف می شود در قالب CTA در SM اجرا خواهد شد.

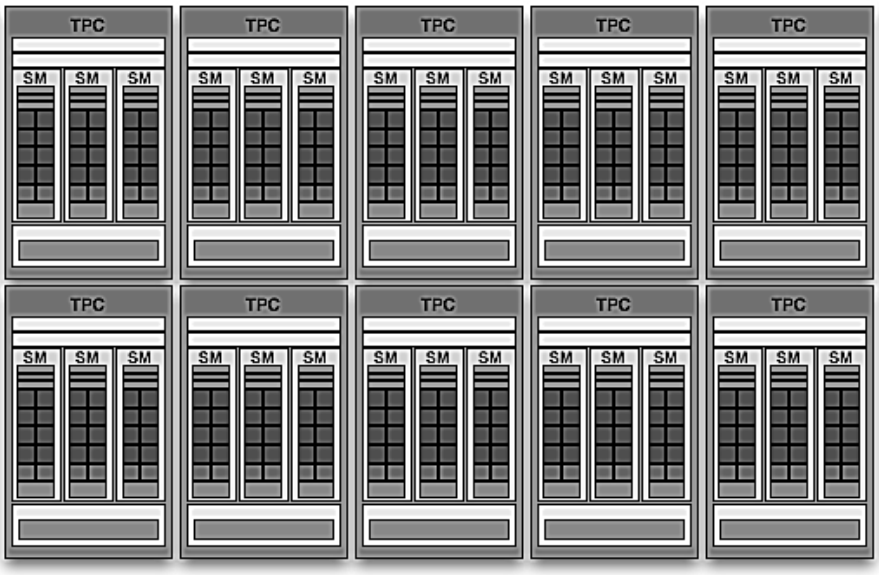
## 2.5 معماری پردازنده های گرافیکی GeForce سری 200

پردازنده‌های گرافیکی ســری 200 دومین نســل پردازنده‌های گرافیکی شــرکت NVIDIA با معماری یکپارچه واحد ســایه زنی هســتند. شــرکت NVIDIA پردازنده‌های گرافیکی ســری 200 را با هدف دســتیابی به اهداف زیر تولید کرد.

- بالا بردن کارایی پردازنده‌های گرافیکی نسل قبل یعنی GeForce سری 8
- بهینه‌سازی معماری برای بازی‌های رایانه‌ای جدید

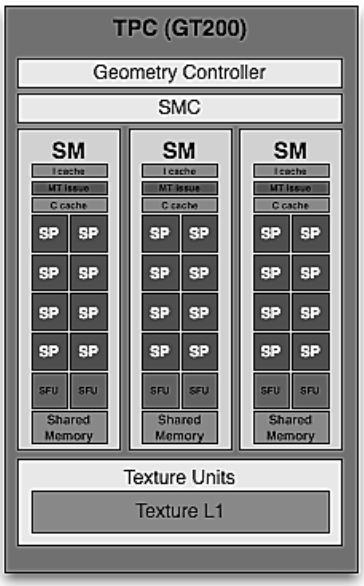
- بهینه‌سازی معماری برای کاهش مصرف انرژی و فضای مصرفی
- ساخت یک پردازنده‌ی گرافیکی قدرتمند برای اجرای پردازش موازی با مقیاس بزرگ‌تر

پردازنده‌ی گرافیکی GeForce GTX 200 اولین پردازنده‌ای است که از نسل دوم واحد سایه زنی یکپارچه شرکت NVIDIA استفاده می‌کند. این پردازنده دارای سرعتی به صورت متوسط 1.5 برابر سرعت پردازنده‌های نسل‌های پیشین است. معماری این پردازنده بر اساس معماری پردازنده‌های گرافیکی GeForce سری هشت است با این تفاوت که معماری این پردازنده در بسیاری از بخش‌ها بهینه‌شده است. در پردازنده‌های سری 200 تعداد TPCها از 8 به 10 عدد در هر پردازنده افزایش پیداکرده است و تعداد SMهای موجود در هر TPC نیز به سه عدد افزایش پیداکرده است. پس با یک حساب سرانگشتی می‌توان متوجه شد که تعداد هسته‌های کودا در این نسل از پردازنده‌ها به عدد 240 رسیده است. در تمامی پردازنده‌های گرافیکی سری 200 نزدیک به 80 درصد از ترانزیستورها برای مصارف پردازشی استفاده شده است درحالی‌که در CPUها به صورت متوسط نزدیک به 20 درصد از فضا برای پردازش اختصاص داده‌شده و حجم زیادی از فضا برای حافظه‌ی کش اختصاص پیداکرده است. از این مقایسه می‌توان نتیجه گرفت که در طراحی CPUها تمرکز بسیار بالایی بر روی کاهش تأخیر وجود دارد؛ انتظار می‌رود سرعت پاسخ یک CPU بسیار بالا باشد. این در حالی است که در مورد پردازنده‌های گرافیکی تمرکز بر روی بالا بردن قدرت پردازش موازی بر روی حجم‌های بسیار زیاد داده‌های ورودی است.




شکل 9-0: معماری GeForce GT 200

در شکل 9-0 معماری GeForce GT 200 نشان داده‌شده است. در بالاترین قسمت واحد برنامه‌ریز ترد[1] وجود دارد که اجرای تردهای مختلف بین TPCها را مدیریت می‌کند. در قسمت پایین هم واحدهای بافت و واحد دستورالعمل‌های اتمیک[2] وجود دارد که نقش آن‌ها مانند نقش همین قسمت‌ها در پردازنده‌های نسل قبل است. همان‌طور که شکل نشان می‌دهد، این سری پردازنده‌ها دارای ده واحد TPC است که اجزای این واحد، با جزئیات در شکل 10-0 نشان داده‌شده است. همان‌طور که در شکل نشان داده‌شده است اجزای واحد TPC مانند اجرای واحد TPC در نسل قبلی پردازنده‌هاست، با این تفاوت که در این نسل تعداد SM بیشتر شده است. پردازنده‌های سری 200 هم مانند نسل قبلی نسبت به تأخیر در فراهم شدن داده مقاوم هستند؛ یعنی درصورتی‌که یک ترد برای دسترسی به داده معطل بماند، سخت‌افزار قادر است بدون تحمیل کردن هیچ‌گونه سرباری مجموعه تردهای دیگری را اجرا کند.




شکل 10-0: اجزای واحد TPC در پردازنده‌های گرافیکی سری 200

یکی از ایرادهای پردازنده‌های سری‌های قبل، این بود که در اجرای برنامه‌های بلند به مشکل پر شدن رجیسترها برخورد می‌کردند. در این نسل از پردازنده‌های گرافیکی اندازه فایل رجیسترها دو برابر شده است که نوشتن برنامه‌هایی طویل با کارایی بالا را تسهیل خواهد کرد. یکی دیگر از مزایای معماری GT 200 این است که واحد محاسبات اعشاری آن محاسبات ضرب جمع را بدون گرد کردن میانی محاسبه می‌کند. در پردازنده‌های قبلی محاسبات MAD در دو مرحله گرد می شد. مرحله‌ی نخست پس از محاسبه‌ی حاصل ضرب بود و مرحله‌ی دوم پس از محاسبه‌ی حاصل جمع. این گرد کردن دومرحله‌ای باعث افت دقت می‌شد. در پردازنده‌های سری 200

[1]Thread Scheduler
[2] Atomic

عملیات MAD روی اعداد اع شاری با دقت م ضاعف، حا صل را پس از عملیات ضرب گرد نکرده و آن را با حفظ تمامی رقم‌های اعشـاری با مقدار ثابت نهایی جمع خواهد کرد. به این روش محاسـبه که نسـبت به روش قبلی از دقت بالاتری برخوردار است Fused Multiply ADD گفته می‌شود.

پردازنده‌های GeForce سری 200 در بـسیاری از جزئیات ازجمله واحدهای بافت، محا سبات هند سی، واحدهای ROC، مدیریت حافظه و مدیریت مصرف انرژی نسبت به نسل‌های پیش از خود برتری داشتند که بیان به‌تفصیل این جزئیات از مو ضوع بحث این کتاب خارج ا ست. پردازنده‌های گرافیکی GeForce سری 200 با توانایی‌های بالایی که داشـــتند پردازش موازی همه‌منظوره با کمک پردازنده‌های گرافیکی را بیش‌ازپیش فراگیر کردند. پس از پردازنده‌های این نسل، شرکت NVIDIA معماری نسل سوم پردازنده‌های گرافیکی با واحد سایه زنی یکپارچه را تحت عنوان فرمی[1] ارائه کرد.

## 2.6 معماری Fermi

زمانی که شرکت NVIDIA شروع به توسعه‌ی معماری فرمی کرد، پردازش موازی با کمک پردازنده‌های گرافیکی دیگر امری نوپا نبود. در آن زمان برنامه‌های بـسیاری برای دو نـسل قبلی از پردازنده‌های گرافیکی تو سعه پیداکرده بود و نظرات بـسیاری از سوی کاربران برای شرکت ار سال شده بود. بدین ترتیب NVIDA توانـست با بهره‌گیری از تجربیات حاصل از پردازنده‌های گرافیکی دو نسل قبل یک معماری قدرتمند ارائه دهد. در این معماری کاستی‌های نسل‌های قبل برطرف شد و استفاده از پردازنده‌ی گرافیکی برای توسعه‌ی برنامه‌های موازی آسان‌تر و بهینه‌تر گردید. مهم‌ترین ویژگی‌هایی که از دیدگاه پردازش موازی در معماری فرمی رعایت شـــده اســـت در ادامه ذکرشده است.

- بهینه‌سازی کارایی واحد محاسبات اعشاری با دقت مضاعف[2]
  - هرچند واحد محاسبات اعشاری با دقت معمولی در پردازنده‌های گرافیکی قبلی دارای کارایی قابل قبولی بود اما برخی از مسـائل مربوط به پردازش‌های موازی نیازمند کارایی بالا در کار با اعداد اعشاری با دقت مضاعف هستند.
- استقرار واحد تشخیص و تصحیح خطای ECC[3]

[1] Fermi
[2] Double Precision Floating Point
[3] Error Correcting Code

- با کمک واحد ECC می‌توان تعداد بسیار زیادی GPU را در یک مرکز پردازش موازی در کنار هم قرارداد و به‌راحتی و بدون نگرانی از بروز خطا بر روی آن‌ها، محا سبات ح ساس به خطا ازجمله محاسبات مربوط به امور مالی را انجام داد.

- کش

- برخی از الگوریتم‌های پردازش موازی نمی‌توانند از حافظه‌ی مشترکی که بر روی پردازنده‌ی گرافیکی در نظر گرفته شده ا ست ا ستفاده کنند به همین دلیل امکان ا ستفاده از کش واقعی نیز در معماری فرمی در نظر گرفته‌شده است.

- افزایش حجم حافظه مشترک[1]

- بر اساس تجربه‌های به‌دست‌آمده از نسل‌های قبلی پردازنده‌های گرافیکی، مشخص‌شده که برخی از امور پردازش موازی نیاز به بیش از 16 کیلوبایت حافظه مشترک دارند تا بتوانند به‌سرعت بالاتری برسند.

- تغییر محتوای سریع

- پردازنده‌هایی که در داخل پردازنده‌ی گرافیکی قرار دارند هم مسئول رسیدگی به امور گرافیکی هستند و هم باید پردازش‌های مربوط به پردازش موازی را بر عهده بگیرند. بدین ترتیب لازم ا ست که بارها کار خود را تغییر دهند. این تغییر کار[2] فرآیندی زمان‌بر ا ست که پردازنده را ملزم می‌سازد تا تمامی محتوای داخلی خود را با محتوای جدیدی تغییر دهد. در معماری فرمی امکان تغییر این محتوا با سرعت بالاتری فراهم‌شده است.

- دستورهای اتمیک سریع‌تر

- بر اساس تجربه‌های قبلی نیاز به دستورهای اتمیک سریع‌تری برای پیاده‌سازی الگوریتم‌های پردازش موازی بر روی پردازنده‌های گرافیکی وجود دا شت. در معماری فرمی این د ستورات نیز گنجانده‌شده است. دستورالعمل‌های اتمیک به دستورالعمل‌هایی گفته می‌شود که اجازه‌ی انجام عملیات بازیابی، پردازش و ذخیره مجدد ق سمت‌های مختلفی از یک ساختمان داده را به‌صورت هم‌زمان به تردهای مختلف می‌دهد. عملیاتی اتمیک نامیده می‌شوند که شامل

[1]Shared Memory
[2]Task Switching

د ستورهای بازیابی، پردازش و ذخیره برای هر ترد به صورت م ستقل و بدون ایجاد مزاحمت با ســایر تردها هســتند. به دلیل افزایش تعداد واحدهای اتمیک در فرمی و وجود کش L2 سرعت عملیات اتمیک در فرمی نسبت به GT200 بیش از بیست برابر بیشتر شده است.

این ویژگی‌های فرمی به علت وجود یک معماری منحصربه‌فرد با اجزای منا سب و متنا سب ا ست. در ادامه برخی از واحدهایی که در بهبود معماری فرمی نقش داشته‌اند توضیح داده خواهد شد.

- افزودن SMهای نسل سوم

  - در نسل سوم SMها تعداد هسته‌های کودا در هر SM به سی‌ودو هسته افزایش پیدا کرد که نسبت به معماری GT200 چهار برابر شده بود. همچنین توانایی محا سبات اعشاری با دقت مضاعف نسبت به پردازنده‌ی گرافیکی نسل قبل 8 برابر افزایش یافت. ازجمله‌ی دیگر بهینه‌سازی‌هایی که در معماری فرمی پدید آمد اضافه شدن قسمت مدیریت وارپ مضاعف بود که به پردازنده این امکان را می‌داد که بتواند د ستورالعمل‌ها را به صورت م ستقل بین دو وارپ مختلف به‌صــورت هم‌زمان توزیع و اجرا نماید. بالاخره اینکه در معماری فرمی 64 کیلوبایت حافظه‌ی رم ا ضافه شد که به صورت پویا قابل تق سیم به دو ق سمت 16 و 48 کیلوبایتی بین کش L1 و حافظه‌ی مشترک بود.

- نسل دوم دستورالعمل‌های موازی اجرای ترد

  - در این نســل فضــای آدرس را برای تطبیق‌پذیری کامل با C++ بهینه کرده‌اند و همچنین واحد محا سبات اعشاری با دقت ساده و م ضاعف ایجاد شد که کاملاً با ا ستاندارد IEEE 754 ســال 2008 میلادی مطابقت دارد. همچنین در این نســل واحد پیش‌بینی انشــعاب[1] ایجادشده است که تأثیر به سزایی در افرایش کارایی پردازنده‌های گرافیکی دارد.

- بهینه‌سازی واحدهای حافظه

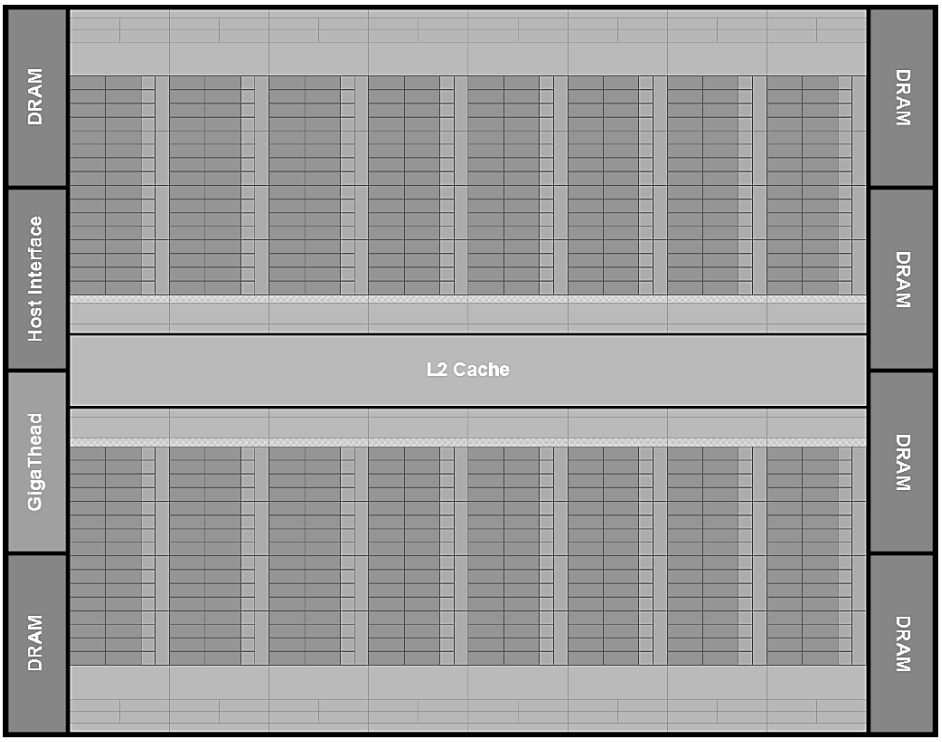
[1] Branch Prediction

- در این نسل از حافظه‌ها سه تغییر مهم رخ داد. نخست افزوده شدن واحد تشخیص و کنترل خطا بود. تغییر دوم بهینه‌سازی ساختار کش و حافظه‌ی مشترک بود و سوم بهینه‌سازی دستورات اتمیک در این نسل بوده است.

- افزودن موتور GigaThread™

- با کمک این موتور، فرمی قادر است با سرعت ده برابر معماری نسل قبلی خودش، تغییر محتوا را انجام دهد. همچنین به کمک این موتور توانایی اجرای هم‌زمان کرنل‌ها و اجرای خارج از نوبت بلاک‌ها فراهم شد.

اولین پردازنده‌ی گرافیکی که بر اساس معماری فرمی طراحی شد دارای سه میلیارد ترانزیستور و 512 هسته‌ی کودا بود. هر هسته‌ی کودا قادر است یک دستورالعمل اعشاری یا صحیح را برای هر ترد در هر پالس ساعت اجرا نماید. این 512 هسته‌ی کودا در شانزده SM که هرکدام دارای 32 هسته‌ی کودا هستند قرارگرفته‌اند. این پردازنده دارای شش حافظه با پهنای باند داده‌ی 64 بیت است و مجموع این حافظه‌ها می‌تواند به 6 گیگابایت حافظه‌ی رم GDDR5 برسد. شکل 11-0 نمونه‌ای از یک پردازنده‌ی گرافیکی با معماری فرمی را نشان می‌دهد.




شکل 11-0: معماری فرمی

همان‌طور که شکل 11-0 نشان می‌دهد حافظه‌ی کش L2 که در وسط قرارگرفته است بین تمامی SMها مشترک است. هرکدام از مستطیل‌های عمودی نشان‌دهنده‌ی یک SM است. نوار پایین نشان‌دهنده‌ی واحد توزیع‌کننده و برنامه‌ریز تردهاست و مستطیل‌های کوچک واحدهای اجرایی هستند.

## 2.6.1 SMهای نسل سوم

نسل سوم SMها دارای ویژگی‌های بارزی است که آن را به‌صورت کامل از معماری نسل دوم جدا می‌کند. هر SM دارای 32 هسته‌ی کوداست. هرکدام از این هسته‌ها دارای یک واحد محاسبات منطقی و ریاضی[1] و یک واحد محاسبات اعشاری[2] است که هردوی آن‌ها دارای خط لوله کامل هستند. در نسل سوم SMها از استاندارد IEEE 754 سال 2008 به صورت کامل پشتیبانی می شود. این پردازنده‌ی گرافیکی دارای یک واحد FMA[3] است که عملیات ضرب-جمع را فقط با یک‌بار گرد کردن پاسخ، محاسبه می‌کند که این باعث بهبود دقت محاسبات می‌گردد. در معماری GT 200 واحد محاسبات اعشاری دارای دقت بیست‌وچهار بیتی بود. در معماری فرمی این دقت به 32 بیت رسید که برای استفاده شدن در زبان‌های برنامه‌نویسی استاندارد مناسب بود. در این

[1] ALU
[2] FPU
[3] Fused Multiply-Add

معماری دقت ALU هم به 64 بیت رسید و تنوع دستورالعمل‌ها افزایش پیدا کرد. این دستورالعمل‌ها شامل عملیات منطقی، شیفت، جابه‌جایی، مقایسه و قراردهی bit-reversed است. هر SM دارای 16 واحد ذخیره و بازیابی اطلاعات است که اجازه می‌دهد عملیات محاسبه‌ی آدرس به‌صورت هم‌زمان برای 16 ترد در هر کلاک محا سبه شود. از طرف دیگر هر SM دارای 4 واحد SFU[1] بوده که قادرند د ستورات سطح بالا را مانند سینوس، کسینوس، محاسبه‌ی معکوس و جذر را اجرا نمایند. هر SFU قادر است یک دستورالعمل را در هر کلاک اجرا نماید. پس دستورات یک وارپ را در هشت کلاک اجرا خواهد کرد. شایان‌ذکر است که تا زمانی که واحدهای محا سبات اع شاری م شغول ا ست امکان ا ستفاده از سایر واحدهای SM وجود دارد. محا سبات اع شاری با دقت مضاعف در بسیاری از مسائلی که نیاز به پردازش موازی دارند وجود دارد در معماری فرمی امکان اجرای 16 دستورالعمل اعشاری با دقت مضاعف در هر کلاک و به ازای هر SM وجود دارد. در معماری فرمی هر SM دو واحد مدیریت وارپ و دو واحد اجرای دستورالعمل دارد که پردازنده‌ی گرافیکی را توانمند می سازد تا بتواند به صورت هم‌زمان دو وارپ را اجرا کند. این واحد دو وارپ را انتخاب می‌کند و از هرکدام از آن‌ها یک دستورالعمل را بر روی 16 هسته، یا 16 واحد ذخیره و بازیابی یا 4 واحد SFU اجرا می‌کند. واضح است که هر دو دستورالعملی نمی‌توانند به‌این‌ترتیب دوتایی باهم اجرا شوند. دو د ستورالعمل صحیح، دو د ستورالعمل اع شاری، یک د ستورالعمل صحیح و یک د ستورالعمل اع شاری، د ستورهای ذخیره و بازیابی و د ستورهای واحد SFU می‌توانند به صورت هم‌زمان اجرا شوند.

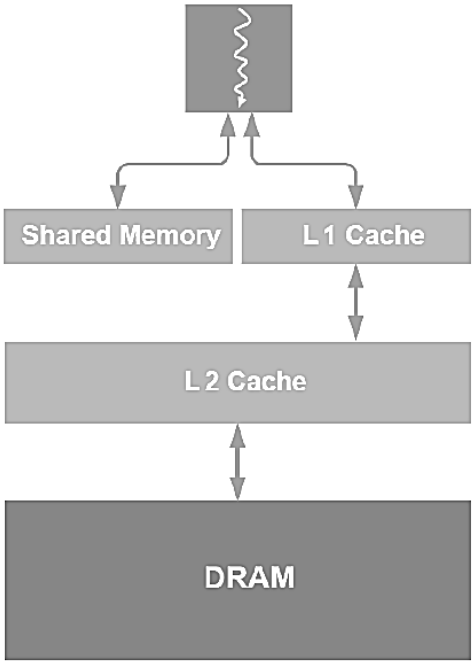
### 2.6.2 حافظه با قابلیت بازپیکربندی

حافظه‌ی م شترک نقش مهمی را در برنامه‌هایی که با پردازنده‌ی گرافیکی نو شته می شوند بازی می‌کند. این حافظه برای برقراری ارتباط بین تردهای مختلف در پردازنده‌ی گرافیکی مورداستفاده قرار می‌گیرد و از همه مهم‌تر نیاز به دسترسی به حافظه‌ی عمومی را کاهش می‌دهد. علیرغم فواید حافظه‌ی مشترک برخی از الگوریتم‌های پردازش موازی قادر نبودند از توانایی‌های این نوع حافظه به اندازه‌ی کافی بهره‌برداری کنند و نیاز به یک حافظه پنهان واقعی دا شتند. این م شکل در پردازنده‌های فرمی حل شد. در فرمی هر SM دارای 64 کیلوبایت حافظه ا ست که این حافظه بنا بر نیاز کاربر می‌تواند به دو قسمت 16 کیلوبایتی و 48 کیلوبایتی تقسیم شود. درصورتی‌که کاربر استفاده‌ی بیشتری از حافظه‌ی مشترک نسبت به کش بکند می‌تواند قسمت 48 کیلوبایتی را به حافظه‌ی مشترک اختصاص داده و قسمت 16 کیلوبایتی را به کش اختصاص دهد و درصورتی‌که کاربر به کش بیش از حافظه‌ی مشترک نیاز داشته باشد می‌تواند جای این دو را باهم عوض کند. شایان‌ذکر است که در معماری GeForce سری

[1] Special Function Unit

8 فضایی که برای کش در نظر گرفته شده بود برابر با 16 کیلوبایت بوده است که در فرمی این فضا هم ازلحاظ حجم و هم ازلحاظ امکان بازپیکربندی بهبودیافته است.

مسئله‌ی دیگر این است که برخی از مسائل نیاز به داشتن یک حافظه‌ی کش دارند و برخی دیگر نیاز به داشتن حافظه‌ی مشترک و برخی هم نیاز دارند هر دو حافظه را داشته باشند. شرکت NVIDIA این مشکل را با ارائه دو سطح کش حل کرده است. در این روش یک کش L2 وجود دارد که بین تمامی MPها مشترک است. این کش که دارای ظرفیتی معادل با 768 کیلوبایت است به SMها این امکان را می‌دهد که به‌جای اینکه اطلاعات را در حافظه‌ی اصلی با یکدیگر مبادله کنند آن‌ها را در کش مبادله کنند و به‌سرعت بالاتری دست پیدا کنند. از طرف دیگر یک کش سطح یک نیز وجود دارد که سایز آن قابل تنظیم است. در حقیقت در این سطح یک حافظه با حجم 64 کیلوبایت وجود دارد که کاربر با توجه به نوع کاربرد می‌تواند آن را به دو قسمت 16 و 48 کیلوبایتی تقسیم کند. درصورتی‌که کاربر نیاز بیشتری به کار با کش احساس کند می‌تواند قسمت 48 کیلوبایتی را به کش اختصاص داده و قسمت 48 کیلوبایتی را برای حافظه‌ی مشترک در نظر بگیرد و بالعکس. لازم به ذکر است که این کش سطح 1 و حافظه‌ی مشترک در هر SM به‌صورت اختصاصی وجود دارد. پس تردهایی که در داخل یک SM قرار دارند نمی‌توانند به حافظه‌ی مشترکی که در داخل SM دیگری قرار دارد دسترسی پیدا کنند و یا اینکه از مزایای کش L1 ای که در داخل SM دیگر است بهره‌مند شوند. اگر بخواهیم به صورت خلاصه بیان کنیم، کش L1 و حافظه‌ی مشترک برای افزایش سرعت و تسهیل ارتباط داخل SMها کمک می‌کنند درحالی‌که کش L2 در افزایش سرعت ارتباط بین SMها کمک می‌کنند.




شکل 12-0: ساختار حافظه در فرمی

## 2.6.3 واحد تصحیح و تشخیص خطا

درصورتی‌که قرار باشد یک پردازنده در مسائل حساس مورداستفاده قرار بگیرد، باید حتماً صحت ثبت و انتقال اطلاعات روی باس‌های مدارهای آن بررسی شود. فرمی اولین پردازنده‌ی گرافیکی است که می‌تواند این بررسی را انجام دهد. فرمی که مجهز به واحد ECC[1] است که قادر است خطای‌هایی را که منجر به تغییر یک بیت شده‌اند را تصحیح و خطاهایی را که منجر به تغییر دو بیت شده‌اند را تشخیص دهد. پس در صورت بروز چنین خطاهایی این امکان فراهم خواهد شد که به‌جای اینکه برنامه با همان داده‌های غلط کار خودش را ادامه دهد متوقف شده و کار را از ابتدا آغاز کند. با توجه به اینکه چنین خطاهایی به علت وجود تشعشعات رادیویی صورت می‌گیرد باید توجه کرد که در کاربردهایی که در آن‌ها امکان وجود چنین خطاهایی وجود دارد از پردازنده‌های گرافیکی استفاده کرد که مجهز به فنّاوری ECC باشند. برای مثال برای استفاده از پردازنده‌ی گرافیکی در یک دستگاه MRI یا یک دیتاسنتر بزرگ باید از پردازنده‌های گرافیکی که دارای چنین قدرتی هستند بهره گرفت. در فرمی، رجیسترها، کش‌های L1 و L2 و حافظه‌ی DRAM دارای فنّاوری تصحیح و تشخیص خطا بوده و همچنین فرمی از استاندارد صنعتی برای کنترل خطا برای نقل‌وانتقال داده بین تراشه‌های مختلف نیز استفاده می‌کند. این بررسی‌های

[1]Error Correcting Code

بروز خطا در قسمت‌های مختلف، پردازنده‌ی فرمی را برای استفاده شدن در سیستم‌های پردازش موازی که نسبت به خطا حساس هستند میسر می‌سازد.

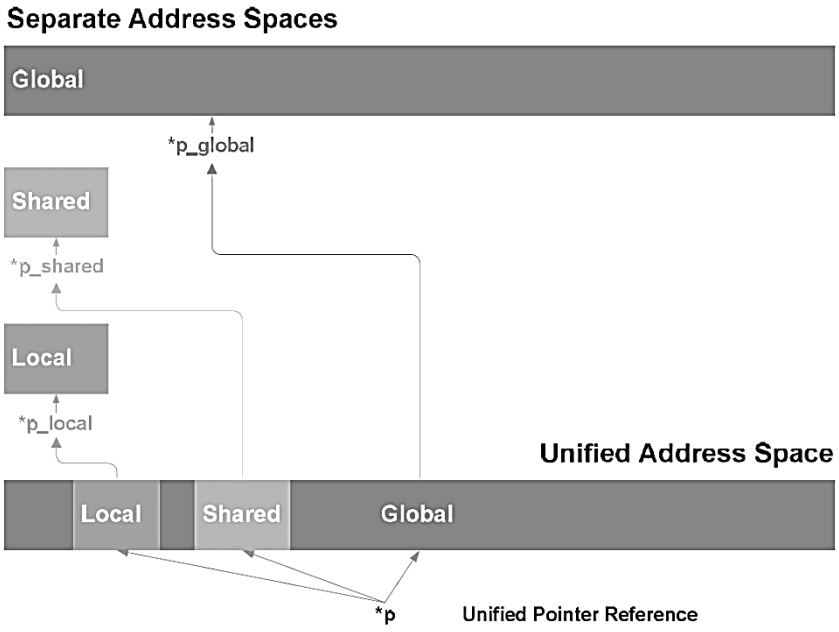
## 2.6.4 تکنولوژی PTX 2.0[1]

فرمی اولین پردازنده‌ای است که از تکنولوژی PTX 2.0 پشتیبانی می‌کند. PTX یک ماشین مجازی سطح پایین است که به‌صورت یک‌لایه‌ی واسط بر روی پردازنده‌ی گرافیکی قرار می‌گیرد. در هنگام اجرای برنامه دستورالعمل‌هایی که برای PTX نوشته شده‌اند توسط درایور پردازنده‌ی گرافیکی به زبان ماشین تبدیل می شوند. علل اصلی که باعث شد تا شرکت NVIDIA از ماشین مجازی PTX استفاده کند عبارت‌اند از:

- امکان ایجاد یک زبان برنامه‌نویسی برای خود PTX که به‌راحتی بتواند علیرغم تغییر نسل‌های مختلف پردازنده‌های گرافیکی بدون تغییر باقی بماند.
- امکان توسعه برنامه با زبان‌های برنامه‌نویسی مستقل از سخت‌افزار مانند C، C++، Fortran بر روی پردازنده‌ی گرافیکی
- فراهم کردن یک زبان برنامه‌نویسی که قادر باشد بر روی پردازنده‌ی گرافیکی که دارای تعداد هسته‌های گوناگونی هستند کار کرده و برنامه را متناسب با نیاز پردازنده‌ی گرافیکی توزیع کنند.

در PTX 2.0 برای نخستین بار فضای آدرس حافظه یکپارچه شد. پیش‌ازاین سه فضای مستقل آدرس حافظه وجود داشت که عبارت‌اند از فضای آدرس حافظه‌ی انحصاری هر ترد، فضای آدرس حافظه‌ی مشترک داخل بلاک‌ها و فضای آدرس حافظه‌ی عمومی. در PTX 1.0 دستورالعمل‌های ذخیره و بازیابی هرکدام از این قسمت‌ها مستقل از سایر قسمت‌ها بود. به‌عبارت‌دیگر می‌بایست در زمان کامپایل کردن برنامه مشخص باشد که مقصد هر دستورالعمل کدام واحد از واحدهای حافظه است. این محدودیت باعث می‌شد که پیاده‌سازی اشاره‌گرهای C و C++ بر روی پردازنده‌های گرافیکی ممکن نباشد زیرا در زمان کامپایل برنامه مقصد اشاره‌گرها مشخص نیست. در فرمی فضای آدرس قسمت‌های مختلف با یکدیگر ادغام شد و نهایتاً یک فضای یکتا ایجاد شده است که از طریق آن می‌توان به تمام قسمت‌های حافظه دست پیدا کرد.

[1] Parallel Thread eXecution (PTX) 2.0




**شکل 13-0: فضای آدرس یکتا در معماری فرمی**

در معماری فرمی قابلیت‌های کار با اعداد اعشـــاری هم بهبود پیدا کرد. در فرمی اعداد غیرمعمولی[1] به‌صـــورت پیش‌فرض توسط سخت‌افزار قابل‌شناسایی شد. همین‌طور چهار روشی که در سند IEEE 754 2008 برای گرد کردن اعداد در نظر گرفته شده است نیز در فرمی قابل پشتیبانی هستند. این چهار روش عبارت‌اند از: گرد کردن به نزدیک‌ترین عدد مجاور، گرد کردن به سمت صفر، گرد کردن به سمت مثبت بی‌نهایت و گرد کردن به سمت منفی بی‌نهایت.اعداد غیرمعمولی اعدادی هســتند که بین صــفر و کمترین عدد غیر صــفر قرار دارند. در نســل‌های قبلی پردازنده‌های گرافیکی این اعداد صفر در نظر گرفته می‌شدند که باعث افت دقت می‌شد. در فرمی این اعداد توسط پردازنده‌های گرافیکی قابل‌پردازش هستند. شایان‌ذکر است که در CPU این اعداد توسط روتین‌های مدیریت خطا بررسی می‌شوند که بررسی آن‌ها زمان نسبتاً زیادی طول می‌کشد.

## 2.6.5 اجرای هم‌زمان کرنل‌ها

در معماری فرمی از اجرای هم‌زمان کرنل‌ها نیز پشتیبانی می‌شود یعنی کرنل‌های مختلف امکان اجراشدن به‌صورت هم‌زمان را دارند. بدین ترتیب در هر زمان امکان اســتفاده از تمام قدرت پردازنده‌ی گرافیکی فراهم شــده و ســرعت نهایی اجرای برنامه بیشتر خواهد شد. این موضوع در شکل 14-0 نشان داده‌شده است.

[1] Sub normal


**شکل 14-0: اجرای هم‌زمان کرنل‌ها در فرمی**

## 2.6.6 کاستی‌ها فرمی

علیرغم کارایی بالای فرمی هنوز هم کاســـتی‌هایی وجود دارد که برای تبدیل کردن پردازنده‌ی گرافیکی به یک ماشین پردازش موازی کارا باید برطرف گردند. برخی از مهم‌ترین این کاستی‌ها عبارت‌اند از:

1- حافظه‌ی نسبتاً کوچک پردازنده‌های گرافیکی

یکی از فرض‌هایی که در تمام سیستم‌های پردازش موازی فوق سریع مطرح می‌گردد این ا ست که برنامه و تمامی اطلاعات موردنیاز آن، همه با هم در حافظه‌ی پردازنده هســـتند. بر اســـاس قانون آمدال که در زمان توســـعه‌ی mainframeها بیان شد در یک سیستم پردازش فوق‌سریع لازم است 1 بایت حافظه و 1 بایت بر ثانیه ورودی و خروجی به ازای هر د ستورالعمل بر ثانیه وجود دا شته با شد پس در سامانه‌ای با توانایی پردازش 500 GFLOPS باید 500 GB حافظه وجود دا شته با شد. هرچند ممکن ا ست بـ سیاری ت صور کنند که این حجم حافظه بیش‌ازحد بزرگ ا ست اما نباید این نکته را فراموش کرد که تو سعه‌ی پردازنده‌های فوق سریع فقط برای حل م سائل د شوارتر است. مسائلی که به دلیل وجود ابعاد وسیع هنوز برای بشر قابل‌حل نیستند.

2- عدم توانایی اتصال مستقیم ورودی و خروجی به پردازنده‌ی گرافیکی

بـ سیاری از برنامه‌های واقعی نیاز دارند که حجم بـ سیار زیادی از داده‌ها را به‌عنوان ورودی دریافت کرده و همین‌طور حجم وســـیعی از داده‌ها را به‌عنوان خروجی بازگردانند. در حال حاضـــر اطلاعات باید از ورودی‌ها و خروجی‌ها گرفته‌شـــده و در اختیار CPU قرار گیرند و ســـپس CPU آن‌ها را در اختیار پردازنده‌های گرافیکی قرار دهد. این انتقال اطلاعات از طریق CPU باعث افت کارایی و سرعت خواهد شد و مسلماً یکی از نقطه‌ضعف‌های سامانه‌های

پردازش موازی ا ست که با کمک پردازنده‌های گرافیکی کار می‌کنند. در صورتی‌که بتوان معماری را به‌گونه‌ای تغییر داد که پردازنده‌های گرافیکی بتواند به‌صورت مستقیم[1] به ورودی‌ها و خروجی‌ها و یا به دیسک سخت دسترسی پیدا کند، سرعت و کارایی پردازنده‌های گرافیکی بهتر خواهد شد.

3- عدم پشتیبانی از حافظه‌های سلسله مراتبی

با توجه به اینکه کاربران مختلف نیاز به سی ستم‌هایی با ابعاد گوناگون دارند، معمولاً سازندگان ترا شه‌های حافظه، آن‌ها را طوری طراحی می‌کنند که امکان قرارگیری دو، چهار و یا حتی هشـــت تا از آن پردازنده‌ها بر روی یک برد ممکن باشد. این استقرار تراشه‌ها بدین سبک به‌آسانی میسر نیست. در درجه‌ی نخست باید یک سخت‌افزار مناسب برای این کار در نظر گرفته شـود که بتواند از چنین سـلسـله مراتبی پشـتیبانی کند. در درجه‌ی دوم باید این ترتیب قرارگیری حافظه‌ها ازلحاظ نرم‌افزاری هم موردبررســـی قرار گیرد و هماهنگی‌های لازم برای امکان کار با چنین تراشه‌هایی فراهم شود. درصورتی‌که پردازنده‌های گرافیکی بتوانند از این سلسله‌مراتب حافظه پشتیبانی کنند، مشکل نخ ست نیز حل خواهد شد زیرا تو سعه‌دهندگان سامانه‌ها قادر خواهند بود با توجه به نیاز سی ستم برای پردازنده‌ی گرافیکی آن، حافظه در نظر بگیرند.

مقایسه‌ی نسل‌های مختلف پردازنده‌های گرافیکی

برای اینکه جمع‌بندی آنچه تاکنون بیان شـد آسـان‌تر گردد در جدول 1-0 جزئیات سـه نسـل اصـلی پردازنده‌های گرافیکی شرکت NVIDIA را با هم مقایسه می‌کنیم.

**جدول 1-0: پردازنده‌های گرافیکی نسل‌های مختلف شرکت NVIDIA**

| GPU | G80 | GT200 | Fermi |
|---|---|---|---|
| تعداد ترانزیستور (میلیون) | 681 | 1400 | 3000 |
| هسته‌های کودا | 128 | 240 | 512 |
| توانایی انجام محاســبات اعشاری با دقت مضاعف | ندارد | 30 FMA ops/clock | FMA ops/clock 256 |

[1] از طریق DMA

| توانایی محاسبات اعشاری با دقت ساده | MADS ops/clock 128 | MADS ops/clock 240 | MADS ops/clock 512 |
|---|---|---|---|
| تعداد SFU | 2 | 2 | 4 |
| تعداد واحدهای مدیریت وارپ در هر SM | 1 | 1 | 2 |
| حافظه مشترک | 16 KB | 16 KB | قابل انتخاب بین 16 KB یا 48 KB |
| کش L1 | ندارد | ندارد | قابل انتخاب بین 16 KB یا 48 KB |
| کش L2 | ندارد | ندارد | 768 KB |
| واحد ECC | ندارد | ندارد | دارد |
| تعداد کرنل‌ها هم‌زمان | 1 | 1 | 16 |
| پهنای باند حافظه | 32 بیت | 32 بیت | 64 بیت |

### 2.6.7 NVIDIA TESLA C2050

پردازنده‌های تسلای سری 20 که بر اساس معماری فرمی طراحی شده‌اند دارای بسیاری از ویژگی‌های ضروری که در سامانه‌های پردازش موازی موردنیاز است هستند. این ویژگی‌ها شامل پشتیبانی از C++، استفاده از فنّاوری ECC و افزایش سرعت 7 برابری در واحد محاسبات اعشاری با دقت مضاعف نسبت به GPUهای تسلای سری 10 هستند.

**جدول 2-0: ویژگی‌های پردازنده‌ی تسلا C2050 و C2070**

| ویژگی | توضیح |
|---|---|
| 448 هسته‌ی کودا | توان ارائه‌ی 515 GFLOPS پردازش به ازای هر GPU |

| | |
|---|---|
| ECC | کنترل خطا بر روی حافظه‌های L1، L2، RAM و ثبات‌ها |
| 6 GB حافظه‌ی DDR5 | توانایی حل بسیاری از مسائل با ابعاد وسیع |
| موتور GIGATHREAD | افزایش سرعت انجام برنامه‌ها با تغییر سریع محتوا |
| انتقال ناهمگام اطلاعات[1] | این GPUها قادر هستند تا زمانی که پردازش قبلی به پایان نرسیده است اطلاعات موردنیاز برای قسمت‌های بعدی را به حافظه بیاورند. به‌عبارت‌دیگر GPU می‌تواند در حین اجرای یک قسمت عملیات انتقال اطلاعات را نیز انجام دهد. |
| پشتیبانی وسیع نرم‌افزاری | می‌توان با کمک C، C++، OpenCL، Fortran و Direct Compute برای کار با تسلا سری 20 برنامه نوشت. از طرفی NVIDIA Parallel Nsight tool نیز برای کاربران Visual Studio فراهم‌شده است. |

[1] Asynchronous Data Transfer

# .3

# فصل سوم

# معماری پاسکال

# معماری پاسکال

در سال 2016 در کنفرانس خبری واقع در شهر san Jose مدل جدیدی از معماری نسل جدید پردازنده های شرکت NVidia را معرفی شد. این معماری که تحت عنوان Pascal معرفی شد با پرچم داری نظیر NVidia Tesla P100 پا به عرصه گشود. بر اساس این کنفرانس و دیتاشیتی که از این معماری برای پردازنده p100 عرضه شد این پردازنده گرافیکی بالاترین کارایی را نسبت به تمام پردازنده های گرافیکی موجود در بازار به ارمغان آورده است. همچنین استفاده آن نوع پردازنده گرافیکی توانسته است بهره ای پردازش های HPC، محاسباتی تکنیکی، یادگیری ژرف و محا سبات با حجم های بالا بر روی بستر های مختلف دیتا سنتر را به حداکثر میزان خود برساند.

در این گزارش سعی بر آن شده است که نیم نگاهی به ویژگی های این معماری تازه وارد و پردازنده گرافیکی مخصوص به آن را شرح داده شود.


## 3.1 Tesla P100: کارایی بی نظیر در کنار ویژگی‌های بارز و چشم نواز برای پردازش های گرافیکی

پردازنده گرافیکی GP100 که از معماری Tesla P100 بهره جویی کرده است ویژگی های مخصوص به خود را دارد که این پردازنده را نسبت به سایر رقبای خود در بازار تمیز داده است. این ویژگی ها عبارت اند از:

عملکری خارق العاده در زمینه های گوناگونی همچون HPC، یادگیری ژرف و ا ستفاده از چندین پردازنده گرافیکی به منظور محاسبات سریع.

NVLink – مدل جدید شـــرکت Nvidia به منظور ارتباط ســـریع با پهنای باند بالا بین چنین کارت پردازنده گرافیکی. این تکنولوژی قادر است بیشترین میزان پردازنده گرافیکی را به هم متصل نماید.

HBM2 – سریعتر، با ظریفیتی بالا همراه با قابلت پشته برای معماری پردازنده های گرافیکی

بهینه سازی معماری یکپارچه و وقفه های بین پردازنده اصلی و گرافیکی که به طور چشمگیری به افزایش سرعت در مدل های برنامه نویسی می انجامد.

تکنولوژی 16 نانومتری FinFet که ویژگی های زیادی از جمله سرعت بالاتر، اشکال زدایی بالا، کاهش انرژی را برای پردازنده گرافیکی به ارمغان می آورد.


**شکل 15: پردازنده ی تسلا P100**

## 3.2 Tesla P100 پیشنهادی عالی برای تکنولوژی NVLink و سرورهای PCI-Express

پردازنده Tesla P100 به دو صورت میتوانند با هم ارتباط برقرار کنند. یک راه سنتی که پردازنده های گرافیکی با اسـتفاده از پورت های PCI-Express به هم و به پردازنده های گرافیکی خارج از سـرور متصـل خواهد شـد. راه دیگیری از ا ستفاده از ماژول SXM2 برای سرور های NVLink میبا شد. پردازنده ی P100 برای سرور های مبتنی بر بســـتر PCI-express این امکان را برای دیتاســـنتر ها فراهم میکند که تمامی پردازند های اصـــلی و گرافیکی با استفاده از این درگاه با هم در ارتباط باشند. همچنین P100 برای سرور های مبتنی بر مدل NVLink این امکان را فراهم میکند که برنامه هایی را به صـــورت توزیع شـــده بر روی چند پردازنده گرافیکی به منظور محاسباتی سنگین همچون یادگیری سریع فراهم آورد

## 3.3 معماری پاسکال GP100: از هر جهتی سریعتر

بنا به رسم قدیم شرکت Nvidia، با هر مدل جدیدی از معماری که معرفی میکند میزان مصرف انرژی را کاهش و عملکرد را افزایش میدهد. این رسم برای معماری پاسکال نیز برقرار بوده. قلب تپنده ای محاسبات پردازنده های گرافیکی را SM ها یا همان Streaming multiprocessor ها تشکیل میدهند. SMها وظیفه ی ساخت، مدیریت، زمانبندی و اجرای عملیات های هر ترد را برعهده دارد.

همانند مدل های قدیمی پردازنده های Tesla، GP100 نیز از آرایه های کلاستر های پردازش گرافیکی (GPC)، SMها و کنترلر های حافظه تشکیل شده است. GP100 توان عملیاتی خودرا وامدار شش GPC که هرکدام از این کلاستر ها شامل 60 SM و همچنین 8 کنترلر 512 بیتی تشکیل شده است. معماری پاسکال عملکرد بهتر خود را تنها وامدار افزایش تعداد SMها نیست بلکه در این معماری هر کدام از SMها به صورت کارامد تری نسبت به مدل های پیشین خود کار میکنند. برای روشن تر شدن این موضوع میتوان به این مورد اشاره کرد که هر SM شامل 64 هسته CUDA و 4 واحد بافتی برای عملیات های کار با تصویر تشکیل شده است. روی هم رفته میتوان به این نتیجه رسید که در این معماری عملا 3840 واحد CUDA و 240 واحد بافتی وجود دارد.


**شکل 16: بلوک دیاگرامی از P100**

بهینه ســازی عملکرد پردازند گرافیکی در محاســبات و همچنین کاهش میزان انرژی به نســبت با این عملکرد دو هدف اصــلی در این معماری پردازنده گرافیکی اســت. میزان تغییراتی که در SMهای این معماری نســبت به مدل قبلی خود یعنی معماری ماکسول صورت گرفت شایان این امر است. شاید بتوان گفت که این عملکرد مدیون نقش تکنولوژی شانزده نانومتری Fin_FET است؛ به طوری که با ارائه این تکنولوژی، به طور نسبی تمامی قطعاتی که از این تکنولوژی بهره بردند به میزان قابل قبولی عملکرد آنها افزایش یافته است.

جدول زیر یک مقایسه پیشرفته P100 با مدل های پیشین Tesla را اورده است :

| Tesla Products | Tesla K40 | Tesla M40 | Tesla P100 (NVLink) | Tesla P100 (PCIe) |
|---|---|---|---|---|
| GPU / Form Factor | Kepler GK110 / PCIe | Maxwell GM200 / PCIe | Pascal GP100 / SXM2 | Pascal GP100 / PCIe |
| SMs | 15 | 24 | 56 | 56 |
| TPCs | 15 | 24 | 28 | 28 |
| FP32 CUDA Cores / SM | 192 | 128 | 64 | 64 |
| FP32 CUDA Cores / GPU | 2880 | 3072 | 3584 | 3584 |
| FP64 CUDA Cores / SM | 64 | 4 | 32 | 32 |
| FP64 CUDA Cores / GPU | 960 | 96 | 1792 | 1792 |
| Base Clock | 745 MHz | 948 MHz | 1328 MHz | 1126 MHz |
| GPU Boost Clock | 810/875 MHz | 1114 MHz | 1480 MHz | 1303 MHz |
| FP32 GFLOPs | 5040 | 6844 | 10608 | 9340 |
| FP64 GFLOPs | 1680 | 213 | 5304 | 4670 |

| | | | | |
|---|---|---|---|---|
| **Texture Units** | 240 | 192 | 224 | 224 |
| **Memory Interface** | 384-bit GDDR5 | 384-bit GDDR5 | 4096-bit HBM2 | 3072-bit HBM2 (12GB)<br>4096-bit HBM2 (16GB) |
| **Memory Bandwidth** | 288 GB/s | 288 GB/s | 732 GB/s | 549 GB/s (12GB)<br>732 GB/s (16GB) |
| **Memory Size** | Up to 12 GB | Up to 24 GB | 16 GB | 12 GB or 16 GB |
| **L2 Cache Size** | 1536 KB | 3072 KB | 4096 KB | 4096 KB |
| **Register File Size / SM** | 256 KB | 256 KB | 256 KB | 256 KB |
| **Register File Size / GPU** | 3840 KB | 6144 KB | 14336 KB | 14336 KB |
| **TDP** | 235 Watts | 250 Watts | 300 Watts | 250 Watts |
| **Transistors** | 7.1 billion | 8 billion | 15.3 billion | 15.3 billion |
| **GPU Die Size** | 551 $mm^2$ | 601 $mm^2$ | 610 $mm^2$ | 610 $mm^2$ |
| **Manufacturing Process** | 28-nm | 28-nm | 16-nm | 16-nm |

## 3.4 Pascal SM

هر واحد SM از پردازنده GP100 شـامل 64 واحد single precision (FP32) اسـت. در مقابل، در معماری ماکسول و یا کپلر که مدلی قدیمی تر نسبت به پاسکال محسوب میشود تعداد این واحد های تک اعشاری به ترتیب 128 و 192 عدد بود. هر واحد SM پردازنده GP100 خود به دو بلاک اصلی پردازشی تقسیم میشود که هر کدام شامل 32 واحد single precision هسته کودا، بافر دستور العمل ها، سیستم مدیریتی وارپ ها و دو واحد توزیع می‌با شند. این درحالی ا ست که پردازنده GP100 نـصف تعداد پردازنده های کودا را نـسبت به معماری ماکسول دارد. ولی با این حال تعداد و حجم رجیسترها در این معماری با معماری پیشین خود برابر است.


**شکل 17: SMهای پردازنده ی تسلا P100**

تعداد رجیسترهای GP100 که در یک SM با مدل های Maxwell GM200 و کپلر GK110 برابر است، ولی مدل GP100 تعداد بیشتری SM نسبت به مدل های پیشگفته دارد فلذا تعداد ریجسترهای این مدل بیشتر از سایر مدل ها خواهد بود. این امر بدین معنا است که هر ترد در پردازنده گرافیکی به تعداد بیشتری ریجستر دسترسی دارد. در یک کلام میتوان گفت که مدل GP100 تعداد بیشتری ترد، وارپ و بلاک را در مقایسه با نسل های قبل خود پشتیبانی میکند.

مقدار حافظه مشترک نیز در مدل GP100 با توجه به تعداد زیاد SM ها افزایش یافته است و نکته دیگر این که پهنای باند حافظه مشترک به طور نسبی دوبرابر نسل پیشین خود شده است. نسبت بالاتر حافظه مشترک، ریجسترها و وارپ ها در هر SM در مدل GP100 این امکان را به کاربر داده است که کد ها را به صورت بهینه به احرا در آورد. به بیانی دیگر تعداد وارپ بیشتری برای هر کنترل کننده عملگری وجود دارد، تعدادی بیشتری عملیات load میتواند صورت بگیرد و همچنین پهنای باند برای ارتباط هر ترد با حافظه مشترک افزایش یافته است. همه اینها باعث افزایش عملکرد پردازنده گرافیکی GP100 شده است .

در مقایسه با معماری kepler، SMهای معماری پاسکال مدل ساده تری از معماری را پیش روی میگیرد به طوری که میزان فضایی که بر روی دای استفاده میکنند کمتر و به مراتب میزان میزان مصرف انرژی نیز برای انتقال داده ها در این مدل کمتر از معماری های پیش از خود خواهد بود. همچنین پا سکال یک مدل مدیریتی برای

افزایش سرعت عملیات های Load و store فراهم کرده که به مراتب عملکرد این د ستور العمل ها را برای واحد های ممیز شناور افزایش داده ا ست. سی ستم مدریت SM ها در معماری پا سکال یک سی ستم هو شمند بوده به طوری که درمقایسه با معماری مکسول عملکردی بهتر را نشان میدهد.

## 3.5 عملکرد بالا در دقت مضاعف

به دلیل اهمیت عملیات های دقت مضاعف در محاسبات تکنیکی و کد های HPC، یکی از هدف های اصلی برای طراحی چنین معماری ای یعنی مغعماری پاسکال افازیش عملکرد عملیات های دقت مضاعف بود. هر SM در پردازنده های GP100 دارای 32 واحد FP64 هستند که نسبت 2 به 1 با واحد های Single Precision دارند. این درحالی ا ست که این ن سبت در مقای سه با پردازنده Kepler GK110 3 به 1 ا ست. فلذا این امر، این امکان را به پردازنده گرافیکی P100 میدهد که عملیات های دقت مضاعف را با استفاده از واحدهای FP64 در حجم کار بالا بهتر کند.

همانند مدل های پیشین پردازنده های گرافیکی، پردازندده GP100 نیز قوانین IEEE 754-2008 را چه برای مدل دقت تکی چه دقت وضاعف پشتیبانی میکند.

پشتیبانی از عملیات ریاضی FP16 برای تسریع در برنامه های یادگیری ژرف:

ر شد روز افزون ر شته هایی از قبیل یادگیری ژرف این خوا سته را به وجود آورده ا ست که یک و سیله محا سباتی سریع به به خصوصی برای این زمینه فراهم آید. یادگیری ژرف ثابت کرده است که میتواند با دقتی بالا و وفق پذیری مناسبی برنامه های کاربری از قبیل توصیف خودکار تصاویر، خودرو های بدون سرنشین، یادگیری زبان های طبیعی و ترجمه ی آنها و از همه مهم تر تولید خودکار هنر های کامپیوتری میتوان نام برد.

برخلاف دیگر برنامه های محا سباتی تکینیکی که نیازمند عملگرد سریع واحدهای اع شاری ا ست، معماری شبکه های عمیق به دلیل متد های backpropagation خطاهایی به طور طبیعی در حین مسیر یادگیری خود دارند. ذخیره سازی داده های FP16 در مقایسه با داده های دقت بالاتری نظیر FP32 و FP64 به نسبت فضای کمتر از حافظه شکبه ع صبی م صرف میکند، که این امر این امکان را به ما میدهد تا شبکه ع صبی بزرگ تری هنگام آموزش شبکه و یا اجرای شبکه طراحی کنیم. استفاده از FP16 عملکرد شبکه عصبی را تا دو برابر نسبت به ا ستفاده از عملیات FP32 بالا برده ا ست و به طور م شابه ا ستفاده از FP16 در انتقال داده ها زمان کمتری نسبت به عملیات های FP32 و FP64 خواهد داشت.

سیستم GP100 SM ISA این امکان را فراهم کرده است که عملیات محاسباتی FP16 را به صورت دوتایی در یک زمان در یک هسته کودا اجرا نماید. گفتنی است که ریجستر های 32 بیتی GP100 میتواند دو مقدار FP16 را در خود جای دهد.

## 3.6 افزایش سرعت عملکرد عملگر های Atomic

عملیات های اتمیک یکی از مهمترین عملیات در برنامه نویسی به صورت موازی است. چرا که به برنامه نویس این امکان را می دهد که چند ترد که به صـورت موازی به اجرا در آمده اند عملیات های خواندن- تغییر- نوشـتن را در یک حافظه مشخص به درستی انجام دهند. معماری کپلر به طور چشمگیری این قابلیت را بهبود بخشید به طوری که این امکان نه تنها بر روی حافظه های محلی میسر بود بلکه میتوان بر روی حافظه های اصلی یا همان global نیز عملیات اتومیک را به اجرا در آورد. اگر چه هر دو معماری فرمی و کپلر هزینه زیادی برای عملیات های اتمیک قائل بودند.

با روی کار آمدن معماری مکسول، بهبود منا سبی برای متغیر های 32 بیتی صحیح فراهم شد به طوری که این عملیات اتمیک به صورت سخت افزاری و کاملا محلی پشتبیانی میشد. همچنین این امکان سخت افزاری برای حالت های 32 بیتی و 64 بیتی برای عملیات مقایسه-تعویض (CAS) برقرار است که این سربار را از برنامه کم خواهد کرد. ( درمقایسه با معماری های فرمی و کپلر که این عملیات به صورت نرم افزاری کنترل میشدند.)

حال پردازنده GP100 با بهبود بخشـــی به عملیات های اتیمک این امکان جمع FP64 را در حافظه global به صورت سخت افزاری فراهم کرده است. تابع atomicAdd در پلتفرم کودا هم در اعداد صحیح و اعشاری 32 بیتی هم 64 بیتی را فراهم کرده اســـت.در مدل های قبل برای جمع های FP64 از حلقه های CAS اســـتفاده میشد که به طور کلی بسیار کند تر مدل های سخت افزاری و محلی است.

قابلیت محاسباتی 6.0

GP100 قابلیت محاســباتی جدید 6.0 را پشـــتیبانی میکند. جدول زیر تفاوتهای پارامترهای مختلف قابلیت های محاسباتی را نشان می دهد.

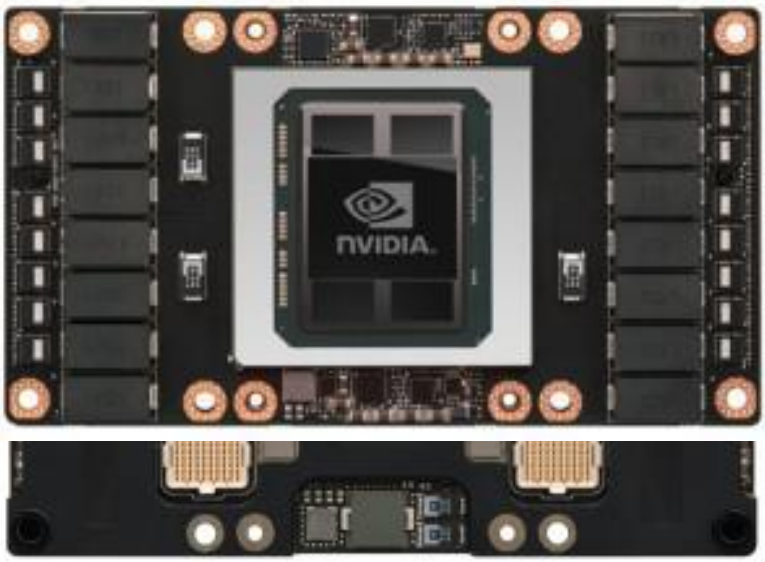
| GPU | Kepler GK110 | Maxwell GM200 | Pascal GP100 |
|---|---|---|---|
| Compute Capability | 3.5 | 5.2 | 6.0 |

| | | | |
|---|---|---|---|
| Threads / Warp | 32 | 32 | 32 |
| Max Warps / Multiprocessor | 64 | 64 | 64 |
| Max Threads / Multiprocessor | 2048 | 2048 | 2048 |
| Max Thread Blocks / Multiprocessor | 16 | 32 | 32 |
| Max 32-bit Registers / SM | 65536 | 65536 | 65536 |
| Max Registers / Block | 65536 | 32768 | 65536 |
| Max Registers / Thread | 255 | 255 | 255 |
| Max Thread Block Size | 1024 | 1024 | 1024 |
| CUDA Cores / SM | 192 | 128 | 64 |
| Shared Memory Size / SM Configurations (bytes) | 16K/32K/48K | 96K | |

## 3.7 بهبودی شگرف در حافظه

بسیاری از برنامه های امروزی به دلیل محدودیت های پهنای باند حافظه دچار مشکل هستند مخصوصا برنامه ها مرتبط با HPC. امروزه، برنامه نویسان برنامه های پردازش سریع بدون توجه به پردازنده ای که با آن کار میکنند، تلاش بسیاری برای بهبود عملکرد برنامه بلاخص برای دسترسی به حافظه دارند. این تلاش ها ممکن است در حیطه نزدیک نگاه داشتن واحد های پرداز شی به سلسله مراتب های حافظه خواهد بود. برخی از برنامه ها از قبیل برنامه های یادگیری سریع که خود شامل چنین لایه از شبکه عصبی است، بابستی با استفاده از یک دیتابیس وسیع و حجیم آموزش داده شود، بزرگترین محدودیتی که دارد، محدودیت حافظه است. بنابرین حافظه ها دو محدودیت بزرگ برای عملیات های محاسباتی دارند: پهنای باند و ظرفیت

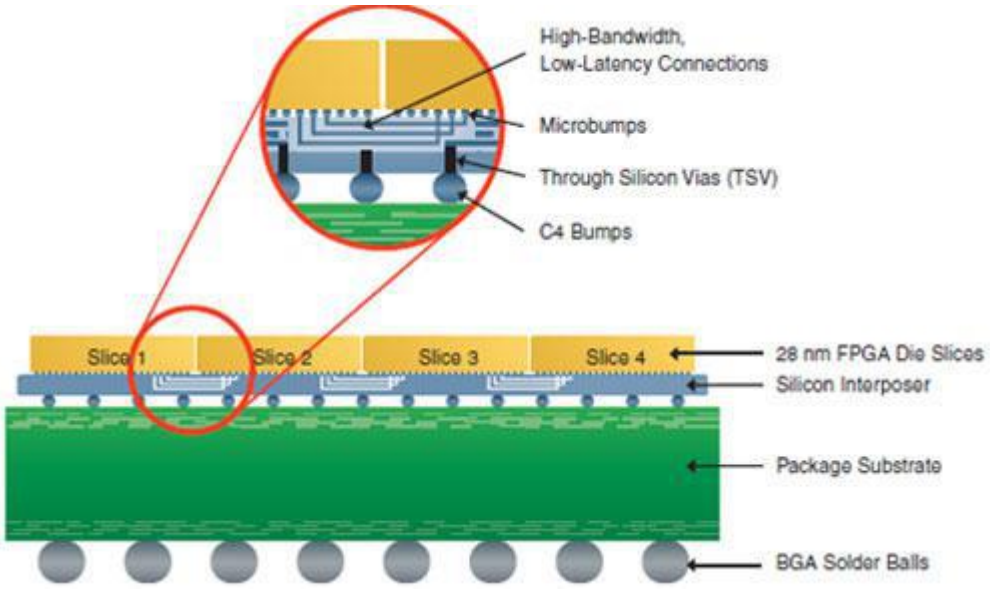
تسـلا P100 هر دوی این محدودیت ها را با اسـتفاده از تکنولوژی حافظه های پشـته ای برطرف کرده اسـت. تکنولوژی ای که چندین لایه از واحد های DRAM را قادر میسـازد تا به صـورت عمودی در یک پکیج درون پردازنده گرافیکی با هم در ارتباط باشـند. تسـلا P100 اولین پردازنده گرافیکی اسـت که با اسـتفاده از تکنولوژی پهنای باند سریع 2 (HBM2) عرضه شده است. این تکنولوژی پهنای باند وسیع و همچنین ظرفیتی بالا، چیزی در حدود 2 برابر، و همچنین بهبود مصرف انرژی را نسبت به GDRR5 دارد.




**شکل 18:شتابدهنده ی P100 از پشت**

بر خلاف GDDR5 که شـامل تکه های جدا از چیپ های حافظه پیرامون پردازنده گرافیکی بود، در تکنولوژی HBM2 شامل یک یا چند پشته عمودی در چند دای مموری خواهد بود. دای های مموری با استفاده از سیم های ریزی که درون–سیکیلونی[1] و microbump ها مرتبط هستند.

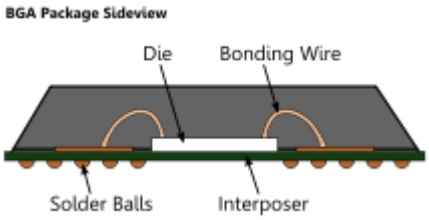
[1] Through-silicon




**شکل 19:نمایشی از شمای دای و میکروبامپ ها**

هر دای 8 گیگابیت HBM2، شـامل 5000 سـوراخ های درون–سـیلیکان می باشـد. در نهایت یک اینترپوز[1] passive از نوع سیلیکان برای مرتبط کردن پشته های حافظه و واحد پردازنده گرافیکی مورد استفاده قرار میگیرد. ترکیب این HBM2ها ، دای پردازنده گرافیکی و اینترپوزهای سیلیکانی در یک پیکیج 55 در 55 میلیمتری تحت عنوان BGA package عرضه می‌شود.

تسلا P100 شامل 4 دای از پشته های HBM2 و 16 گیگابایت از حافظه است. این پردازنده قادر است با پهنای باند 720 GB/s ارتباط بر قرار کند ( گفتنی اسـت که این پهنای باند در حالت پیک میباشـد) به طوری که این پهنای باند سه برابر بیشتر از مدل تسلا M40 است.




**شکل 20: محل اینترپوزر**

[1] Interposer

## 3.8 ECC Memory

از دیگر مزیت های تکنولوژی HBM2 میتوان به پشتیبانی محلی از رفع خطاهای کد[1] اشاره کرد. این ویژگی قابل اطمینان بودن عملیات های محاسباتی برای برنامه هایی که به خرابی داده ها حساس هستند فراهم میکند. به عنوان مثال میتوان به برنامه هایی که در کلاستر های عظیم و یا سوپر کامپیوتر ها که پردازنده گرافیکی حجم بالایی از از دیتاست ها را پردازش میکنند اشاره کرد.

تکنولوژی ECC یک بیت از خطا را قبل از آن که به سیستم آسیبی بزند شناسایی و رفع می‌کند. در مدل GDDR5 تکنولوژی ECC را برای محافظت از محتوای حافظه به صورت محلی پشتیبانی نمیکند، این رفع یابی بایستی در درگاه داده صورت پذیرد که خود موجب به کاهش سرعت می‌شود.

در مدل پردازنده گرافیکی Kepler GK110 امکان حفاظت از داده های حافظه با استفاده از تکنولوژی ECC به صورت محدود می‌دهد. این محدودیت بدین معنا ست که جزئی از حافظه از این قابلیت استفاده میکند. 6.25% از کل حافظه GDDR5 به منظور بررسی ECC رزور شده است. در حالتی که برای پردازنده K40 حافظه 12 گیگابایتی را در نظر بگیریم، 750 مگابایت تنها سهم یان تکنولوژی شده و مابقی حافظه یعنی 11.25 گیگابایت بایستی بر روی درگاه داده عیب یابی شوند. همچنین دسترسی به بیت های ECC باعث کاهش خفیفی در پهنای باند حافظه نسبت به عدم استفاده از ECC خواهد بود. بنابراین HBM2 به دلیل آن که ECC را به صورت محلی و سخت افزاری پشتیبانی میکند، دیگر این کاهش پهنای باند برای این معماری بی تاثیر خواهد بود. همچنین ECC در هر زمانی فعال خواهد بود. همانند پردازنده گرافیکی GK110، پردازنده GP100 نیز از رفع خطای تکی[2] و شناسای خطای دوتایی[3] (SECDED) پشتیبانی میکند.

## 3.9 NVLink برای ارتباط سریع

NVLink یک تکنولوژی جدید و سریع به منظور ارتباطات بین پردازنده های گرافیکی است که توسط شرکت Nvidia معرفی شده است. با توجه به پشتیبانی ای که پردازندهد P100 از SXM-2 میکند، تکنولوژی NVLink به این امکان را دارد که پردازنده های گرافیکی با هم و یا با پردازنده اصلی به صورت بسیار کارامد ارتباط برقرار کنند.

---

[1] Error correct code (ECC)

[2] Single error correct

[3] Double error correct

امروزه، استفاده از چند پردازنده گرافیکی در یک ایستگاه‌های پردازشی[1] همانند استفاده از چندین node در کلاستر های HPC و یا سیستم های آموزش به یادگیری های ژرف مرسوم است. ارتباطات قوی در یک سیستم چند پردازنده ای بسیار ضروری است. هدف از معرفی تکنولوژی NVLink ارائه یک سیستم ارتباطی بین پردازنده های گرافیکی است که بتواند با پهنای باند بالا با استفاده از درگاه های PCI-Express نسل سوم با هم در ارتباط باشند. از دیگر اهداف این تکنولوژی سازگاری آن با GPU ISA به منظور پشتیبانی از حافظه های محلی پردازنده های گرافیکی در سیستم های چند پردازنده ای است.

پشتیبانی از GPU ISA بدین معنا است که بتوان با استفاده از تکنولوژی NVLink پردازنده های گرافیکی را به هم متصل کند و برنامه ما به طور مستقیم بر روی داده هایی که در چند پردازنده گرافیکی قرار دارد اجرا شود. پردازنده گرافیکی همچنین میتواند عملیات اتومیک حافظه ای را بر روی پردازنده های گرافیکی کنترلی از راه دور را بر عهده گیرد که این امر باعث بهبود در توزیع مناسب برنامه ها می‌شود.

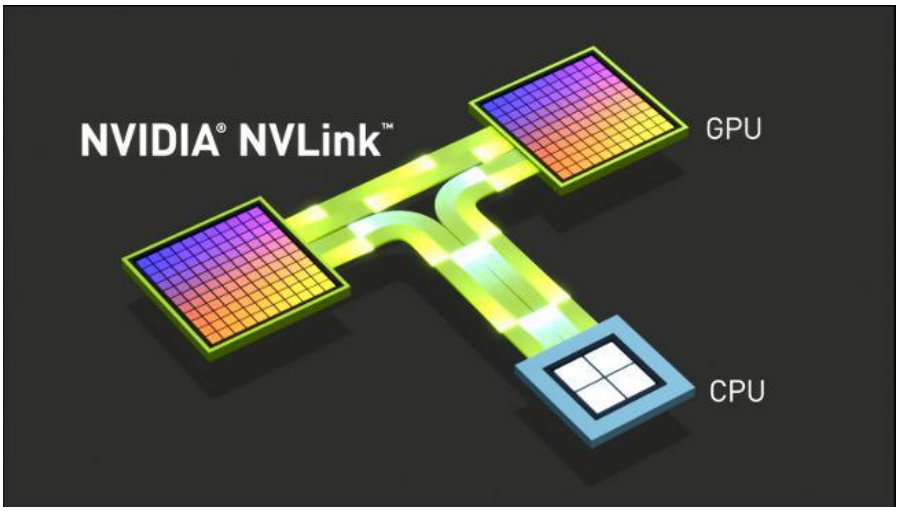
NVLink از ارتباطات سریع سیگانلی انویدیا[2] بهره می‌جوید. NVHS داده ها را بر روی جفت های تفاضلی که با سرعتی حداکثر 20 گیگابیت بر ثانیه دارد انتقال میدهد. هشت عدد از این جفت های تفاضلی یک sub-link را تشکیل میدهد که داده را در به صورت یک طرفه انتقال میدهد و دو sub-link – هر کدام برای یک جهت– یک لینک را تشکیل میده که این لینک دو پردازنده گرافیکی یا یک پردازنده گرافیکی با یک پردازنده اصلی وصل میکند. یک لینک می تواند تا 40 گیگابیت بر ثانیه را به صورت دو طرفه انتقال دهد. چند لینک میتواند با هم متصل شوند و گنگ[3] را تشکلی دهد که پهنای باند وسیع تری را برای ارتباط بین پردازنده ها فراهم آورد.
NVLink ای که در پردازنده P100 به کار رفته است از چهار لینک تشکلی شده است. این بدان معنا است که بیشینه پهنای باندی که به صورت تئوری قابل انتقال به صورت دوطرفه خواهد بود برابر است با 160 گیگابیت در ثانیه.

---

[1] Workstation

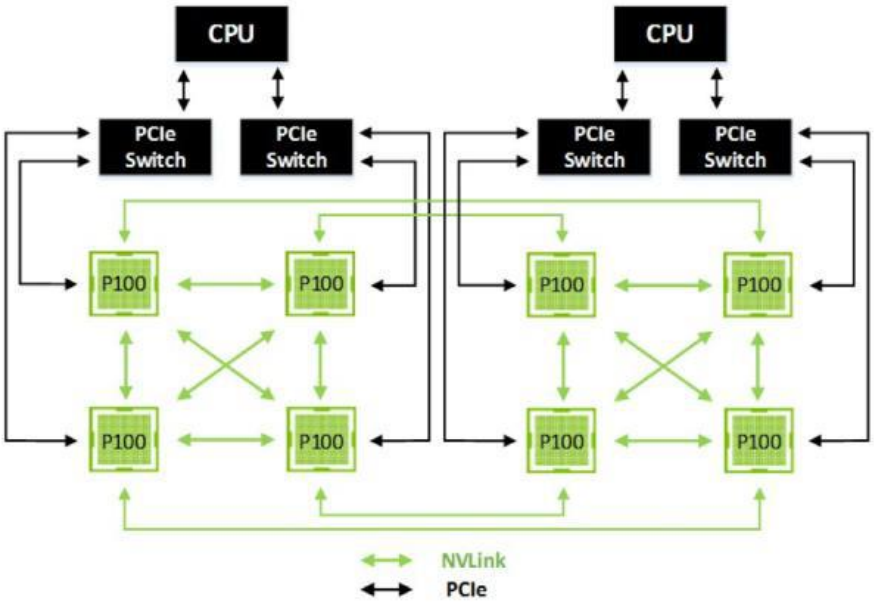
[2] Nvidia’s High-speed Signaling interconnect (NVHS)

[3] gangs




**شکل 21: NVLink**

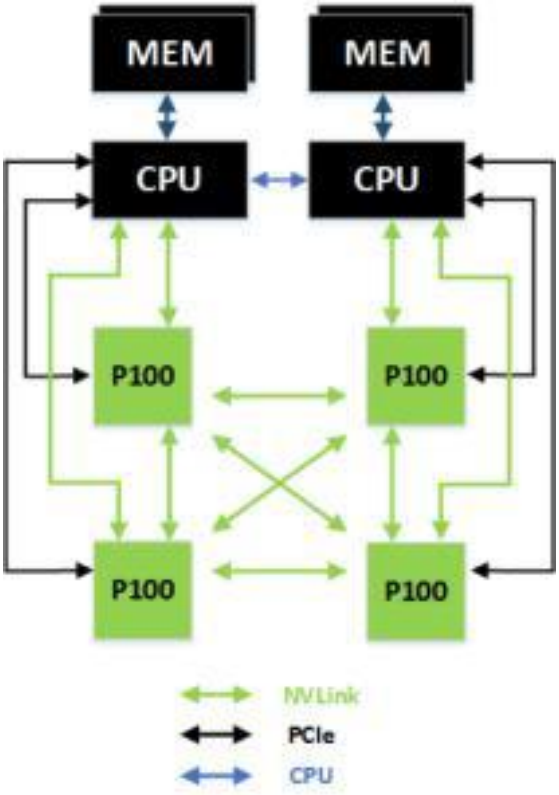
تصویر زیر 8 پردازنده گرافیکی به صورت مش های مکعبی ترکیبی[1] که 4 تا از آنها به صورت کامل با استفاده از NVLink به هم مرتبط هستند و یک NVLink نیز بین این دسته های چهارتایی متصل است نشان میدهد. هر پردازنده گرافیکی در دسته چهارتایی خود به پردازنده اصلی دسته خود به طور مستقیم با استفاده از درگاه PCI-E متصل است.




**شکل 22: هشت پردازنده گرافیکی که به صورت hyper cube قرار دارند**

[1] Hybrid cube mesh

در حالی که وظیفه اصـلی NVLink وصـل کردن چندین پردازنده گرافیکی GP100 به یکدیگر اسـت، این امکان را که پردازنده گرافیکی GP100 را به پردازنده های اصـــلی IBM متصـــل نمود، وجود دارد. تصـویر زیر معرف این موضـوع اسـت که چهار پردازنده گرافیکی با اسـتفاده از تکنولوژی NVLink به CPUها متصل شده است. در این پیکربندی، هر پردازنده گرافیکی مجموعا دارای سرعت 120 گیگابیت بر ثانیه به صـورت دو طرفه با 3 پردازنده گرافیکی دیگر اسـت و 40 گیگابیت ارتباط دوطرفه با پردازنده اصلی است.




**شکل 23: 4 پردازنده گرافیکی که با استفاده از تکنولوژی NVLink به پردازنده اصلی و خود متصل اند.**

## 3.10 افزایش بهره وری برنامه نویسی با استفاده از حافظه یکپارچه[1]

حافظه یکپارچه یکی از اصلی ترین ویژگی های برنامه نویسی به زبان کودا است به طور یکه عملیات برنامه نویسی را به طور چشـــمگیری برا برنامه نویس راحت میکند. این بدان معناســـت که میتوان برنامه و اطالاعات لازم را به

[1] Unified memory

پردازنده گرافیکی فرستاد ولی در عین حال از یک حافظه به منظور دسترسی پردازنده گرافیکی و پردازنده اصلی به حافظه استفاده نمود. مدل پاسکال GP100 با ارتقای سطح عملکرد حافظه یکپارچه پیشرفت شگرفتی در این امر انجام داده است.

با معرفی کودا 6، حافظه های یکپارچه نیز معرفی شـــده که در آن یک حافظه که هم از طریق پردازنده گرافیکی و هم پردادزنده اصلی قابل دسترس است. این حافظه ها از دو پردازنده اصلی و گرافیکی با استفاده از یک اشاره گر قابل دسترس بود. سیستم کودا به صورت خودکار داده‌هایی که در حافظه یکپارچه قرار دارد را بنا به نیاز به حافظه اصلی یا گرافیکی منتقل میکند، بدین صورت که گویی پردازنده اصلی برنامه ای را در حافظه اصلی خودش پردازش میکند و یا پردازشگر گرافیکی داده را در حافظه گرافیکی خود دارد.

پلتفرم کودا 6 برای حافظه های یکپارچه با محدودیت هایی روبه‌رو بود. یکی از این محدودیت ها آن بود که اگر سیستم کنترل حافظه تماسی با پردازنده اصلی داشته باشد باید عملیات همزمان سازی قبل از اجرای کرنل ها صورت پذیرد. همچنین امکان دستر سی همزمان پردازنده گرافیکی و پردازنده اصلی به یک نقطه از حافظه امکان پذیر نبود. از دیگر محدودیت ها نیز میتوان به محدود بودن حافظه یکپارچه به اندازه حافظه پردازنده گرافیکی اشاره کرد.

**CPU Code**

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

**CUDA 6 Code with Unified Memory**

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

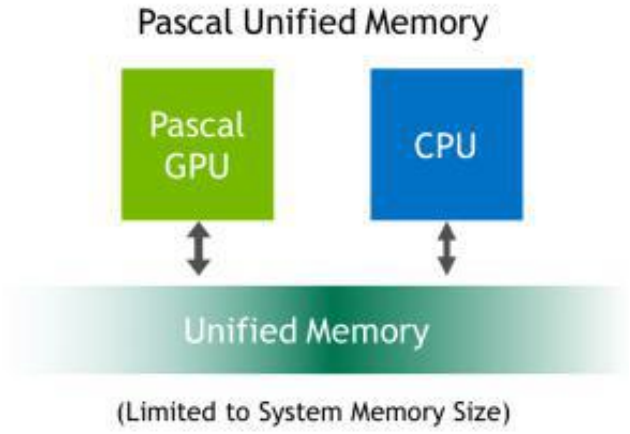
**شکل 24: نمونه کد کودا 6**

### 3.10.1 حافظه یکپارچه پردازنده P100

با توجه به مزیت هایی که حافظه یکپارچه کودا 6 برای کاربر به ارمقان آورد، در مدل پاســـکال GP100 تعداد بیشتری از ویژگی برای راحتی برنامه نویسان و ارتباطات بین حافظه‌های پردازنده های گرافیکی و اصلی نمو پیدا

کرد، به طوری که ارتباط CPU برای اجرای برنامه های موازی شده با پردازنده گرافیکی با سرعت بالایی ارتقا پیدا کرد. دو ویژگی سخت افزاری مهم عامل این پیشرفت ها بوده اند: پشتیبانی از آدرس دهی به میزان بالایی از حافظه و رفع یابی آدرس دهی صفحه ای.

GP100 امکان آدرس دهی GPU را تا 49 بیت برای آدرس دهی مجازی فعال کرد. این میزان عظیم از آدرس دهی به منظور پوشـاندن حافظه های اصـلی نسـل جدید، همپا با پردازنده های گرافیکی جدید لازم بود. بنابراین، حافظه یکپارچه GP100 قابلیت دسترسی به تمام آدرس های پردازنده گرافیکی و پردازنده اصلی دارد.




پ شتیبانی پردازنده گرافیکی از خطایابی آدرس دهی صفحه یکی از بنیادی ترین تغییراتی ا ست که در این ن سل از پردازنده گرافیکی اتفاق افتاده است. ترکیب ویژگی های آدرس دهی مجازی 49 بیتی و عیبابی آدرس دهی صفحه مزایای فراوانی را برای کاربران فراهم کرده ا ست. اول از همه با ید ستی گفت که عیبیابی صفحه بدین معنا ست که پلتفرم کودا دیگر نیازی به همزمان سازی سی ستم مدریت حافظه ای قبل از اجرای کرنل‌ها ندارد. اگر کرنلی که بر روی پردازنده گرافیکی در حال اجراست به حافظه ای از پردازنده گرافیکی دسترسی پیدا کند که درون حافظه نباشد، خطا رخ خواهد داد و این امکان فراهم می‌شـود که صـفحه مورد نظر به صـورت خودکار به درون حافظه پردازنده گرافیکی انتقال یابد. متناوبا، ممکن اسـت که صـفحه مورد نظر به درون حافظه پردازنده گرافیکی نگاشـته شـود به طوری که ارتبط پردازنده گرافیکی با داده مورد نظر از طریق درگاه PCI-Express و یا NVLink صورت پذیرد. گفتنی است که به طور کلی نگاشت[1] سرعت عمل بالاتری نسبت به انتقال داده درون پردازنده گرافیکی دارد.

فلذا با توجه به این سی ستم خطایابی صفحه در ن سل جدید پردازنده های گرافیکی و کودا 8، وا ستگی داده های global و حافظه ی یکپارچه تضـمین خواهد شـد. این بدان معناسـت که با پردازنده GP100، میتوان به صـورت همزمان پردازنده های گرافیکی و پردازنده های اصلی به حافظه یکپارچه دسترسی داشته باشند. گفتنی است که این

---

[1] mapping

امر در مدل های کپلر و مکسـول قابل قبول نبود. به این دلیل که هیچ تضـمینی وجود نداشـت، زمانی که پردازنده گرافیکی در حال اجرای کرنلی باشد، پردازنده اصلی قادر به دسترسی به حافظه یکپارچه داشته باشد.

در نهایت، با پشتیبانی از این ویژگی، هر نوع اختصاص دهی حافظه توسط سیستم عامل از قبیل malloc و یا new هم از طریق پردازنده گرافیکی و هم از طریق پردازنده اصلی توسط یک اشاره گر قابل درسترس اسـت. ( می توانید به عکس زیر مراجعه کنید. در این سـیسـتم، اسـتفاده از حافظه یکپارچه به صـورت پی شفرض مورد ا ستفاد ها ست. با این ح ساب دیگر نیازی به یک تخ صیص دهنده خاص برای ا ستفاده تخصیص حافظه نخواهد بود.

CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

Pascal Unified Memory*

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  free(data);
}
```

*with operating system support

این نکته نیز نبای ست فراموش شود که چندین د ستورعمل خاص به منظور فعال سازی این متد بر روی سی ستم عامل نیازمند اسـت که شـرکت NVidia با همکاری Red Hat و انجمن های لینوکس این عملگر قوی را فعال کرده اند.

## 3.11 NVidia DGX-1 یادگیری ژرف با استفاده از سوپرکامپیوترها

دانشـمندانی که در حوزه پردازش اطلاعات و یا هوش مصـنوعی کار میکنند، به دقت بالا، سـادگی و در عین حال سرعت بالا برای سی ستم های یادگیری ژرف نیازمندند. آموزش هر چه سریع تر با شد، نمو خلاقیت های پیرو این حیطه از علم و همچنین فروش محصولات وابسته به آن در بازار جهانی سریع تر خواهد بود. پرداز شگر NVidia DGX-1 اولین سیستمی است که به منظور پردازش های ژرف طراحی شده است، به طوری که یک پک کامل از سـخت افزار ها و نرم افزار های لازم به منظور اجرای سـریع و آسـان الگوریتم های لازم را فراهم آورد. این

سوپرکامپیوتر انقلابی در سیستم یادگیری ژرف محسوب میشود به طوری که عملکرد سیستم تا 170 TFLOP/s برای عملیات های FP16 ارتقا پیدا کرد.

NVidia DGX-1 اولین سروری است که از پردازنده گرافیکی تسلا P100 که استفاده کرده است. این سرور قادر است که 8 پردازنده گرافیکی P100 را پیکربندی کند.

**شکل 25: یک سوپرکامپیوتر DGX-1**

پیکربندی 8 پردازنده گرافیکی P100 با استفاده از تکنولوژی NVLink این امکان را فراهم میسازد که این 8 پردازشگر را به صورت 4 تای به صورت کامل به هم مرتبط کند. بدین معنا که 4 پردازشگر به صورت دوبه‌دو به هم متصل هستند. هر پردازشگر در این سیستم چهارتایی با استفاده از درگاه PCI-E متصل بوده و از آن به پرداز شگر اصلی متصل خواهد بود. گفتنی است که در بدترین حالت، استفاده از سرور NVidia DGX-1 در حالتی که از 8 پرداز شگر P100 استفاده کند، دوازده برابر افزایش سرعت در حالتی از نسخه قبلی پردازشگر ها استفاده شود، دارد.

| | DUAL XEON | DGX-1 |
|---|---|---|
| FLOPS (CPU + GPU) | 3 TF | 170 TF |
| AGGREGATE NODE BW | 76 GB/s | 768 GB/s |
| ALEXNET TRAIN TIME | 150 HOURS | 2 HOURS |
| # NODES FOR 2HR TAT | >250* | 1 |

# .4

# فصل چهارم

# تسریع الگوریتم های یادگیری ژرف و هوش مصنوعی با استفاده از پردازنده گرافیکی

# تسریع الگوریتم های یادگیری ژرف و هوش مصنوعی با استفاده از پردازنده گرافیکی

هدف ا صلی از علم هایی نظیر یادگیری ژرف و هوش م صنوعی بر آورده کردن نیاز هایی از قبیل ساخت یک ما شین که بتواند به صورت هو شمندانه به ا سان خدمت کند. این هو شمندی بدین صورت ا ست که نیازی به یک دســتور عمل خاص و یا الگوریتم های مشــخص نداشــته و ماشــین به صــورت خودکار با یادگیری از پیرامون خود نیازهای ب شری را ار ضا نماید. یادگیری ژرف یکی از مهمترین د ست آورد های جدید هوش مصنوعی است که در سال های اخیر رونق گرفته است.

یادگیری ژرف این امکان به مغز، خواه انسان و یا خواه ماشین، می دهد که با یادگیری از محیط پیرامون، ت صمیماتی در ست و به جا در موقعیت های خاص بگیرد. اما ازعان به این نکته ضروری ا ست که آموزش به چنین ما شینی نیاز به حجم عظیمی از داده دارد. به علاوه، در مدل های یادگیری ژرف که در آن ها از لایه های زیادی ا ستفاده شده ا ست، پردازش این اطلاعات را برای کاربر سخت و طاقت فر سا میکند. در سال 2012، شرکت گوگل، پروژه ای تحت عنوان google's Deep Learning را که تحت مغز گوگل شناخته میشود راه اندازه کرد. این پروژه به منظور شناسایی گربه ها با استفاده از فیلم های یوتوب کلید زده شد. اما این مغز مصنوعی نیازمند حداقل 2000 پردازنده اصلی و یا به عبارتی دیگر نیازمند 16000 ه سته CPU، به منظور قدرت دادن به این پردازش ها بود. کمتر کمپانی ای در دنیا وجود دارد که بتواند چنین حجم عظیمی از منابع و ســخت افزار را یکجا برای انجام چنین پردازشــی فراهم آورد. در همین هنگام، شرکت NVidia با همکاری با دان شگاه Stanford ت صمیم به ا ستفاده از پردازنده های گرافیکی برای پردازش های مورد نیاز الگوریتم های یادگیری ژرف گرفتند. نتیجه ی چنین ت صمیمی، چیزی ن شد جز آن که 12 پردازنده گرافیکی از شــرکت NVidia توانایی یادگیری ژرف همگام با 2000 پردازنده اصــلی را برآورد کرد. همانطور که از اعداد مشــخص اســت این کاهش میزان پردازنده، نه تنها در میزان م صرف انرژی صرفه جویی ب سزایی کرد، بلکه این امکان را به دیگر شرکت ها داد تا با تهیه تعدادی که پردازنده گرافیکی کمتر از تعداد انگشــتان دســت، به رقابت با پردازنده های Intel در الگوریتم های یادگیری ژرف بپردازند.

خیلی ها بر این باورند که انقلاب یادگیری ژرف به ســال 2012 که در این ســال مســابقه ImageNet با حظور سردم داران این علم یعنی Krizhevsky,sutskever و Hilton برگزار شد. این اشخاص با به کار

گیری پردازنده های گرافیکی، الگوریتم CNN را که بدان تحت عنواه AlexNet نیز یاد میشـــود را راه اندازی کردند و نتیجه چیزی نشـد جز آن که عملکردی دوچندان بهتر نسـبت به رقبایی که از کلاسـتر های قوی برای این مسابقه استفاده کرده بودند، به دست آورد. لازم به ذکر است که کریژوسکی و تیمش هیچ کدی در زمینه پردازش ت صویر ننو شتند. بلکه متد آنها بدین صورت بود که با ا ستفاده از یادگیری ژرف این امکان را با ماشـین دادند تا خودش شـروع به یادگیری کند. آنها با طراحی یک شـبکه عصـبی موسـوم به AlexNet و آموزش آن با اسـتفاده از میلیون ها مثال که خود نیازمند تریلیون ها محاسـبات ریاضیست این مهم را به انجام رساندند. شایان ذکر است که شبکه عصبی AlexNet بهترین کد به دست انسان نوشته شده در آن سال شد.

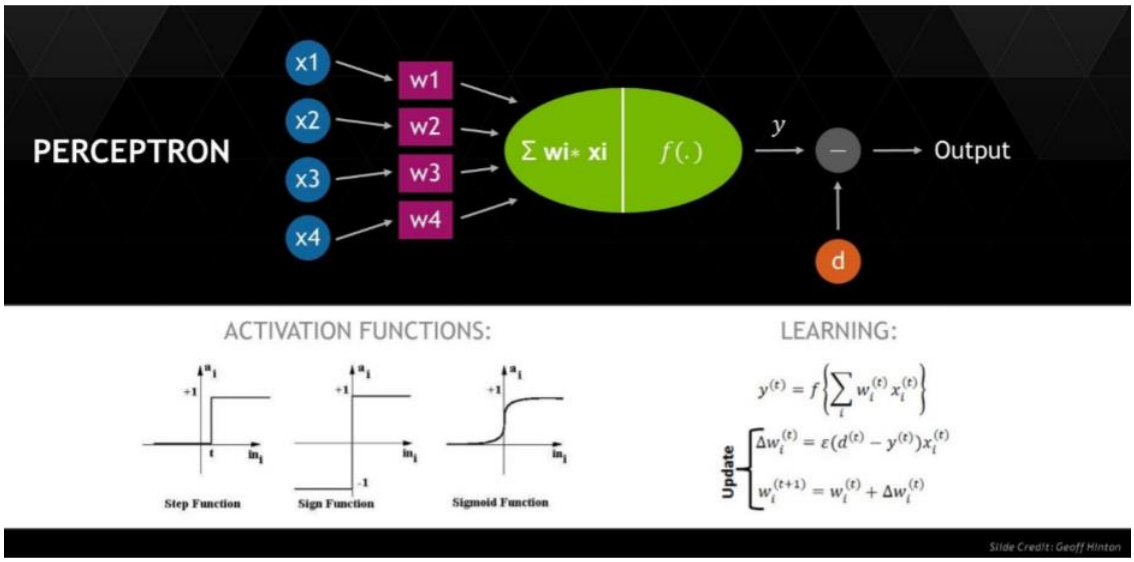
پس از آن، رفته رفته، شـــبکه های ژرف عصـــبی بر روی پردازنده های گرافیکی انقلابی را در زمینه های پردازش تصـــویر به طور خاص و به طور کلی الگوریتم های وابســـته به یادگیری راه انداختند. کاربرد این علم به مراتب فراتر از درک ماسـت، به عنوان مثال اسـتفاده از ماشـین هایی که بدون سـرنشـین قادر به رانندگی بر روی هر سـطحی و تحت هر شـرایطی هسـتند و یا مترجم هایی که بلادرنگ هر زبانی هر با زبان دلخواه ما ترجمه سلیس میکند. امروزه، ا ستفاده از یادگیری ما شین در کنار پردازنده های گرافیکی تمام امکانات فوق الذکر را برای ما فراهم کرده اسـت. به طوری که اسـتفاده از پردازنده های گرافیکی به طور چشمگیری مرحله زمان آموزش شبکه عصبی را کاهش داده است.

## 4.1 یادگیری ژرف در نگاه کلی

یادگیری ژرف یک تکنیکی اسـت که مغز اسـنان را با اسـتفاده از شـبکه ها عصـبی و متدهای یادگیری ع صبی شبیه سازی می کند. با یادگیری بی شتر، ما شین باهوش تر خواهد شد و در نتیجه جواب های دقیق تری را برای ما فراهم خواهد کرد. به عنوان مثال، کودکی را در نظر بگیرید که توسط پدر و مادرش در حال یادگیری مطالبی از قبیل شناسایی دقیق اشکال هندسی است؛ به تدریج این امکان برای کودک فراهم خواهد شد که ا شکال هند سی مختلفی را بدون نیاز به والدین خود شنا سایی کند. به طور م شابه، یادگیری ژرف و یا یادگیری عصبی این امکان را به سیستم میدهد که اشیا را از تصاویر شناسایی و آن ها را د سته بندی کند. این قدرت میتواند فراتر از مثال های ساده رود به طوری که سیستم هر شکلی خواه کامل باشد یا ناقص خواه واضح باشد یا پوشیده شناسایی نماید.

در سطحی ساده و مبتدی، عصب ها در مغز انسان به صورت یک هاب پذیرای ورودی های مختلف است، در نظر بگیرید. و مرتبه اهمیت هر کردام از این ورودی ها به مراتب مشـخص شـده اسـت. در نهایت نیز خروجی یک عصب نیز به عنوان ورودی به عصبی دیگر منتقل خواهد شد تا تصمیمات لازم انجام شود.

عصب، همانطور که در شکل زیر مشخص است، یک مدل بسیار ساده شده از مغز انسان است. با توجه به شکل، هر پرسپترون[1] چندین ورودی که هر کدوم مشخص کننده یک ویژگی از داده ورودی است، وجود دارد. وظیفه نهایی این سلسله پر ستپرون ها آموختن و شنا سایی و طبقه بندی سازی هر کدام از داده های ورودی ای به هنگام تست الگوریتم میباشد. گفتنی است که هر کدام از این ویژگی ها که به ورودی پر سپترون داده شده ا ست یک وزن به خ صوص که میزان اهمیت آن ویژگی را م شخص میکند ن شان میدهد.




**شکل 26: یک پرسپترون که مدلی ساده از شبکه عصبی است**

برای مثال، یک پر سپترون را در نظر بگیرید که هدف آن شنا سایی شماره صفر در عکس های د ست نو شته ا ست. واظهاً، عدد صفر به صورت های گوناگونی میتواند نو شته شود که این صور به د ست خط نگارنده وابسته است. حال پرسپترون در این مرحله عکسی که شماره صفر بر آن است را به عنوان ورودی میگیرد. سپس آن را به بخش های مختلفی تق سیم میکند. به عنوان مثال این عکس میتواند به 4 بخش که مشـخص کننده 4 ویژگی اسـت تقسـیم می‌شـود. منحنی قسـمت راسـت بالایی به ویژگی شـماره 1

[1] Perceptron

نگاشت میوشد منحنی پایین نیز به ویژگی دوم نگاشت خواهد شد و این روند به همین صورت ادامه پیدا میکند. وزنی که به هر کدام از این ویژگی ها تخصیص می‌یابد مشخص کننده میزان اهمیت هر کدام از این ویژگی هاست. قسمت بیضی شکل سبز رنگ در شکل بالا مشخص کننده مرحله است که پرسپترون هر کدام از وزن ویژگی ها را متناسب با ورودی آن مورد محاسبه قرار میدهد که در نهایت تشخیص دهد آیا عدد صفر است یا خیر.

مهم ترین ویژگی شبکه عصبی آموزش شبکه برای رسیدن به جوابی بهتر است. هر پرسپترون برای شناسایی عدد صفر از عکس، در ابتدا به صورت تصادفی مقدار دهی میشوند و سپس با متدهای یادگیری شروع به ارتقای ماتریس وزن ( که ارزش های هر کدام از ویژگی های تصویر را مشخص می کند) میکند. این پرسپترون که برای عدد صفر تخصیص داده شده است، روند را ادامه میدهد تا با این نتیجه برسد که آیا عکس عدد صفر را نشان میدهد یا خیر. این فاز برنامه را نشر رو به جلو[1] می گویند. چنانچه شبکه عصبی قادر به شناسایی درست عدد نباشد، باستی علت این عدم توانایی شناسایی شود، این شناسایی عدم توانایی معمولا با مقدار بزرگی خطا مشخص میشود. برای برون رفت از این موضوع بایستی ماتریس وزن برای ویژگی ها رو طوری تغییر داد که در نهایت شبکه عصبی قادر به شنا سایی دقیق اعداد با شد. این پرو سه از خطای نا شی از جواب فیدبک گرفته و بنا به این فیدبک ماتریس را د ستخوش تغییر داده. این روش را نشر بازپس خطا[2] گویند.

به طور کلی، پرسپترون یک مدل ساده از شکبه عصبی است. این در حالی است که مدل های چند لایه از شبکه عصبی بر اساس همین مدل ساده پرسپترون ساخته شده است. هنگامی که شبکه یادگیری لازم را برای شنا سایی و طبقه بندی در ست ا شیا را انجام داد، برنامه حا ضر به اجرا خواهد بود و به صورت برنامه اجرایی ( دیگر نیازی به یادگیری دوباره نیست ) به اجرا خواهد آمد. از مثال های کارامد متد های DNN[3] میتوان به شناسایی اعداد در دستخط ها برای استفاده از چک های بانگی در دستگاه های خودپرداز و یا شنا سایی عکس های دو ستان در سامانه امنیتی فیسبوک، پیشنهاد فیلم بر ا ساس 50

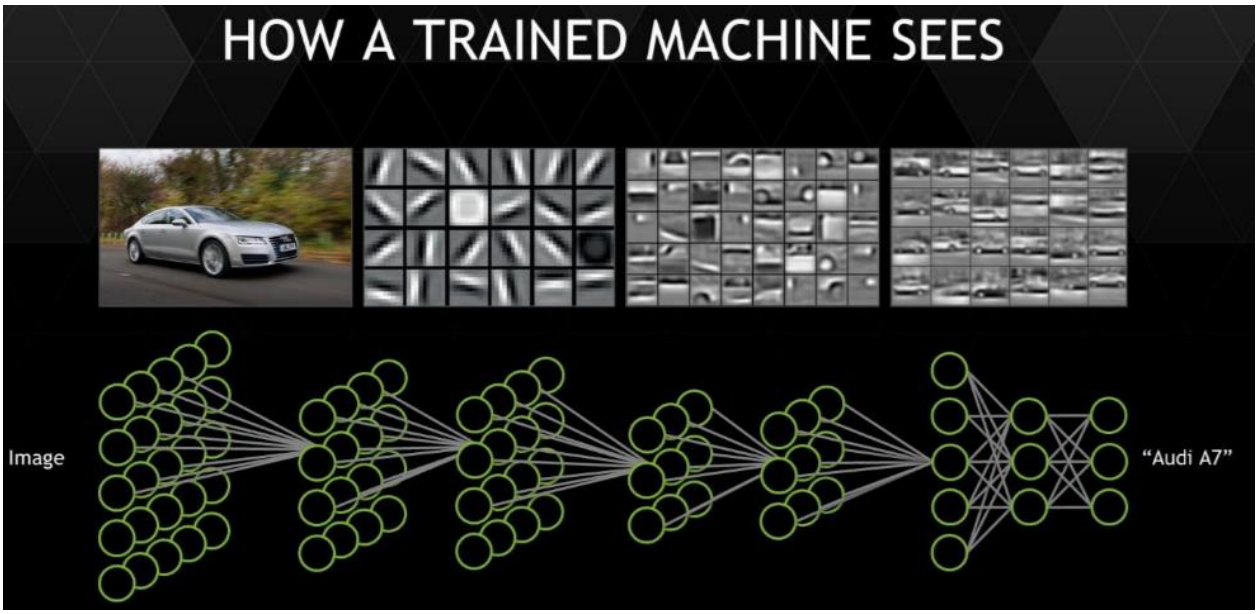
[1] Forward propagation

[2] Backward propagation

[3] Deep neural network

میلیون کاربر شـــبکه اینترنتی نتفلیکس[1]، شـــناســـایی و طبقه بندی انواع مختلف خودروها، پیاده ها و خطرهای جاده برای ماشین های بدون راننده و در نهایت ترجمه گفتار انسان به صورت بلادرنگ.

یک مدل چندلایه شبکه عصبی همانطور که در شکل زیر مشخص است، ممکن است از چند لایه که گره های آن به یکدگیر مت صل ه ستند، ت شکیل شده با شد؛ هر کدام از این گره ها م شخص کننده ی یک ویژگی ورودی است که خروجی هر کدام از این گره ها به عنوان ورودی چند گره دیگر در لایه های بعدی باشد.




**شکل 27: یک مدل پیچیده از شبکه عصبی که نیازمند محاسبات سنگین است**

در مدل شـکل بالا، اولین لایه، از شـبکه عصـبی، ویژگی های اصـلی از عکس را به قسـمت های مختلف می شکند تا الگوهای اولیه از ت صویر مانند خط ها و زووایا به د ست آید. پس از آن در لایه دوم این با سر هم سازی این خطوط و زوایا الگویی پیچیده تر نسبت به لایه قبل مانند چرخ ها، شیشه ها و برف پاکن ها را می سازد و در لایه بعدی مدل خودرو را م شخص میکند و در نهایت در لایه آخر این شبکه ع صبی برند خودرو را مشخص میکند. که در این مثال خودرو آوودی[2] A7 بود.

[1] Netflix

[2] Audi

یک راه دیگر به جای استفاده از شبکه کامل وصل[1] استفاده از لایه های کانولوشنی[2] است. یک عصب در یک شبکه کانولوشنی به یک عصب دیگر تنها از طریق یک ناحیه نسبتا کوچکتر از مدل پیشگفته متصل خواهد بود. به طور کلی، این ناحیه ممکن است از یک شبکه 5 در 5 اعصاب تشکیل شده باشد ( شاید هم 7در7 و یا 11 در 11). اندازه این شبکه، اندازه فیلتر ما را مشخص میکند. بنابرین یک لایه کانولوشنی عملا به عنوان یک تابع کانولوشن برای داده ورودی نقش ایفا میکند. این نوع از مدل شبکه کانولوشنی یک مدل ساده شده از مدل واقعی مغز وافع در شبکیه گانگلونی[3] است که عمدتا وظیفه کورتکس دیداری را برعهده دارد.

در شبکه کانولوشنی DNN، ماتریس وزن فیلترها برای لایه های مختلف از شبکه عصبی مانند هم هستند. به دیگر سخن، یک شبکه کانولوشنی همانند یک مدل زیرلایه وار که هر کدام از فیلتر های مختلفی ساخته شده اند پیاده سازی می شوند صدها فیلتر مختلف ممکن است برای لایه کانولوشنی به کار روند. این متد ممکن است تا جایی پیش رود که یک لایه کانولوشنی از چندصد تابع کانولوشنی استفاده کند و نتیجه همه‌ی آنها را به لایه بعدی خود به عنوان ورودی دهد. شبکه های DNN که به صورت لایه های کانولوشنی به کار رفته اند را عمومال شبکه های عصبی کانولوشنی[4] گویند.

## 4.2 پردازنده های گرافیکی NVidia: موتوری قوی برای یادگیری ژرف

با روی کار آمدن شبکه های عصبی ژرف و یا شبکه های کانولوشنی این سوال پیش آمد که محاسبات قوی و طولانی برای یادگیری این شبکه از طریق روش پس انتشار خطا چگونه میخواهد انجام پذیرد. به علاوه، شبکه های عصبی ژرف نیازمند دیتاست های حجیم برای فرایند یادگیری هستند تا بتوانند در مرحله ی تست جوابی با دقت بالا در اختیار کاربر قرار دهند. این بدان معناتس که هزار میلیون از نمونه های ورودی وجود دارد که بایستی هم از طریق روش انتشار رو به جلو و هم روش پس انتشار خطا مورد آزمایش قرار گیرند.

---

[1] Fully connected

[2] Convolutional layer

[3] Retinal ganglion

[4] Convolutional Neural Networks (CNNs)

امروزه دیگر این امر مشخص است که یادگیری این شبکه ها چه در زمینه های صنعتی و چه در زمینه های آموزشی با استفاده از پردازنده های گرافیکی صورت میپذیرد، چرا که این پردازنده ها نسبت به مدل های همتای خود یعنی پردازنده های اصلی (CPU) هم از لحاظ مصرف انرژی و هم از لحاظ سرعت اجرای برنامه ها به صورت موازی بسیار قوی تر و مقرون به صرفه تر هستند. برای روشن تر شدن این موضوع، لازم است گفته شود که شبکه های عصبی خود از مقداری زیادی عصب تشکیل شده اند و این اعصاب طبیعتا به صورت موازی با هم در ارتباطند. این همکار موازی عصب ها با یکدگیر یک نگاشت ساده بر روی پردازنده های گرافیکی برای ما پدید می آورد که به مراتب سرعتی بالاتر در آموزش این شبکه ها نسبت به مدل سنتی یعنی استفاده از CPU در اختیار ما قرار میدهد.

شبکه های عصبی به صورت کلی بر اساس ماتریس هایی با ابعاد بزرگ و محاسبات سنگین در چند لایه ای سوار ده اند که محسابات لازم برای این شبکه ها عصبی خود نیازمند پردازنده های قوی با قابلیت محاسبات ممیز اعشاری با پهنای باند های وسیع و سریع است. پردازنده های گرافیکی با تعدد هسته هایی محاسباتی ای که دارند، این محاسبات ماتریسی سنگین را برای ما بهینه کرده اند به طوری که کارایی در حدود چندصد TFLOPS در اختیار ما قرار داده اند.

در حال حاظر شرکت NVidia سردمدار پردازنده های گرافیکی است که به صورت اختصاصی معماری هایی برای شبکه های عصبی ژرف و هوش مصنوعی تولید میکند. پردازنده های گرافیکی شرکت NVidia به این شبکه های عصبی شتاب لازم را می‌دهند که برنامه های خود را با سرعت های ده تا بیست برابری به اجرا در آورند. این کار ها عموما با کاهش زمان یادگیری برنامه از چندین هفته و چندین روز به تنها چندین ساعت است. با همکاری بسیاری از متخصصین در زمینه های مختلف، این امکان برای پردازنده های گرافیکی این شرکت به وجود آمده است که طراحی پردازنده های گرافیکی، معماری این سیستم ها، کامپایلر و الگوریتم های مورد نیاز را ارتقا ببخشند. تا سه سال قبل، پردازنده های گرافیکی شرکت NVidia کمک های شایانی را برای بهبود سرعت شبکه های یادگیری ژرف کرده اند که این رشد نمایی گویا قرار بر توقف به خود نخواهد گرفت.

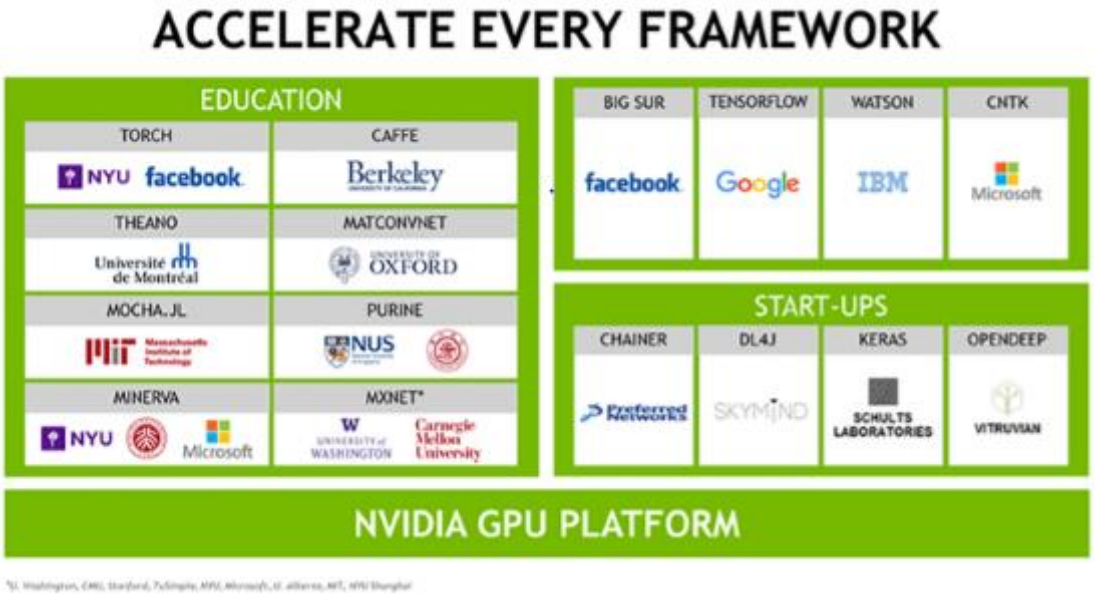
## 4.3 نرم افزار ها و toolkit های لازم برای یادگیری ژرف

همانطور که پیشتر گفته شد، هوش مصنوعی، یکی از سردم داران علم های به روز دنیاست. برنامه نویسی سخت در این زمینه برای برنامه نویسین و توسعه دهندگان این علم یکی از محدودیت های بزرگ است. برنامه نویسی نسبتا آسان و غنی پلتفرم کودا این امکان را به برنامه نویسان و محققان داده است که سرعت پیشرفت این علم را بیشتر کنند. شرکت NVidia برنامه ها و کتاب خوانه های مختلفی از قبیل NVIDA DIGIT، cuDNN، cuBLAS و دیگر برنامه ها را برای اپلیکیشن های یادگیری ماشین به منظور اجرا بر روی بستر های ابری، دیتا سنترها، و workstation ها و یا حتی در سیستم های embedded و نهفته تحت عنوان Deep Learning software kit (SDK) فراهم کرده است. هدف اصلی برنامه نویسان آن است که بتوانند برنامه ای بنویسند که هرجایی که خواستند بتوانند آن را به اجرا در آورند. پردازنده های گرافیکی NVidia در نقطه از جهان وجود دارد. چه در سیستم های شخصی خانگی، چه در لپتاپ ها، سرورها و یا در سوپر کامپیوتر ها؛ و حتی در سیستم های مبتنی بر بستر ابری همچون آمازون[1]، گوگل، IBM، فیسبوک، بایدو[2] و ماکروسافت. تمامی فریموورک های مبتنی بر هوش مصنوعی مبتنی بر پردازنده های گرافیکی NVidia هستند، از شرکت های اینترنتی گرفته تا استارتاپ های و موسسات تحقیقاتی. شرکت NVidia به منظور هر مصرفی از خانگی گرفته تا سیستم های سوپرکامپیوتر پردازنده گرافیکی خاص خود را تولید مینماید؛ برای مثال برای مصارف خانگی GeForce را و یا برای سیستم های ابری و یا سوپرکامپیوتر ها معماری Tesla را معرفی کرده است. برای ربات ها و یا پهباد ها نیز معماری Jetson که برای سیستم های نهفته کاربرد دارد را معرفی نموده است.

---

[1] Amazon

[2] Baidu




**شکل 28: فریم وورک های مختلف برای کار با یادگیری ژرف**

# فصل چهارم

# جمع‌بندی و نتیجه‌گیری و پیشنهادات

## جمع‌بندی و نتیجه‌گیری

به طور کلی میتوان گفته که پردازنده های گرافیکی شرکت NVidia با ر شد فزاینده ای که این سال ها در زمینه های گوناگون داشـــته اند، به بزرگترین پرچم دار ســـخت افزار های الگوریتم های یادگیری ژرف تبدیل شده اند. شرکت NVidia با تمرکزی خاص بر روی الگورتیم های یادگیری ژرف، هوش مصنوعی و شبکه های عصبی این امکان را به موسسات مختلف و دانشگاه ها داده است که فرایند یادگیری شبکه های عصبی را با سرعتی چند برابر حالت قبل به اجرا در آورند. گفتنی است که شرکت NVidia با ارائه نســـل جدید از معماری پردازنده گرافیکی تحت عنوان پاســـکال در را برای دیگر پردازش های موازی نیز بازتر کرد و این امکان را به کاربران داده اســت که کار با اعداد اعشـــاری را با ســـرعتی دو برابر نســـل قبلی یعنی کپلر به اجرا در آورد. این شرکت، با ارائه تکنولوژی جدید NVLink از محدودهایی که درگاه PCI-E خلاص شده و با سرعتی 4 برابر این درگاه، ارتباطات بین پردازنده های گرافیکی را برقرار ساخته است. از دیگر امکاناتی که این معماری همراه با پلتفرم کودا 8 ارائه کرده اســـت، نســـل جدیدی از حافظه های یکپارچه میبا شد که کاربر را از تخ صیص دهی حافظه به صورت تخ ص صی رهایی میبخ شد و عملا این امکان را به حافظه اصـــلی و حافظه گرافیکی میدهد که با هم و در یک زمان به یک نقطه از حافظه د ستر سی دا شته با شند. گفتنی ا ست که این حالت در معماری های قبلی اگر رخ میداد، باعث خطا در واکشی داده ها از حافظه شده و برنامه خطا نشان می‌داد.

## پیشنهادات

پردازنده های گرافیکی NVidia به منظور جلب ر ضایت برنامه نویسان نیازمند پلتفرمی کاربردو ستدارانه تر ا ست، چرا که هر برنامه‌ای که به زبان کودا نو شته می شود تنها مختص به معماری ای ا ست که برنامه نویس برای آن نوشـــته اســـت، میباشـــد. فلذا این پلتفرم باید این امکان را برای کاربران و برنامه نویســـان فراهم کند که فارغ از معماری فعال در ســـیســـتم، در بهینه ترین حالت ممکن، برنامه به اجرا در آید. از دیگر پیشنهاداتی که به این معماری گرافیکی میتواند وارد کرد، عدم اجرای برنامه هایی ا ست که بر روی معماری های ا صلی اجرا می شوند. این پردازنده هر چند به منظور کارهای گرافیکی طراحی شده ا ست ولی در آینده ای نه چندان دور میتواند جایگزینی مناسب برای پردازنده های اصلی باشد.

# منابع و مراجع


[1] دیتاشیت معماری GP100 شرکت NVidia

[2] دیتاشیت معماری jetron از شرکت NVidia.

[3] Professional CUDA C Programming
Book by John Cheng, Max Grossman, and Ty McKercher

[4] CUDA for Engineers: An Introduction to High-Performance Parallel Computing
Duane Storti, Mete Yurtoglu