

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده 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 |

[فصل اول مقدمه 1](#_Toc475140617)

[فصل دوم معماری پردازنده‌های گرافیکی 3](#_Toc475140618)

[2‌.1‌ کارتهای گرافیک 4](#_Toc475140619)

[2‌.2‌ معماریCUDA 7](#_Toc475140620)

[2‌.3‌ حافظه در کودا 12](#_Toc475140621)

[2‌.4‌ معماری پردازنده های گرافیکی GeFroce سری 8 15](#_Toc475140622)

[2‌.5‌ معماری پردازنده های گرافیکی GeForce سری 200 24](#_Toc475140623)

[2‌.6‌ معماری Fermi 27](#_Toc475140624)

[2‌.6‌.1‌ SMهای نسل سوم 31](#_Toc475140625)

[2‌.6‌.2‌ حافظه با قابلیت بازپیکربندی 32](#_Toc475140626)

[2‌.6‌.3‌ واحد تصحیح و تشخیص خطا 34](#_Toc475140627)

[2‌.6‌.4‌ تکنولوژی PTX 2.0 35](#_Toc475140628)

[2‌.6‌.5‌ اجرای همزمان کرنل ها 36](#_Toc475140629)

[2‌.6‌.6‌ کاستیها فرمی 37](#_Toc475140630)

[2‌.6‌.7‌ NVIDIA TESLA C2050 39](#_Toc475140631)

[3. فصل سوم معماری پاسکال 41](#_Toc475140632)

[3‌.1‌ Tesla P100: کارایی بی نظیر در کنار ویژگی‌های بارز و چشم نواز برای پردازش های گرافیکی 42](#_Toc475140633)

[3‌.2‌ Tesla P100 پیشنهادی عالی برای تکنولوژی NVLink و سرورهای PCI-Express 43](#_Toc475140634)

[3‌.3‌ معماری پاسکال GP100: از هر جهتی سریعتر 44](#_Toc475140635)

[3‌.4‌ Pascal SM 46](#_Toc475140636)

[3‌.5‌ عملکرد بالا در دقت مضاعف 48](#_Toc475140637)

[3‌.6‌ افزایش سرعت عملکرد عملگر های Atomic 49](#_Toc475140638)

[3‌.7‌ بهبودی شگرف در حافظه 50](#_Toc475140639)

[3‌.8‌ ECC Memory 53](#_Toc475140640)

[3‌.9‌ NVLink برای ارتباط سریع 53](#_Toc475140641)

[3‌.10‌ افزایش بهره وری برنامه نویسی با استفاده از حافظه یکپارچه 56](#_Toc475140642)

[3‌.10‌.1‌ حافظه یکپارچه پردازنده P100 57](#_Toc475140643)

[3‌.11‌ NVidia DGX-1 یادگیری ژرف با استفاده از سوپرکامیپوترها 59](#_Toc475140644)

[4. فصل چهارم تسریع الگوریتم های یادگیری ژرف و هوش مصنوعی با استفاده از پردازنده گرافیکی 61](#_Toc475140645)

[4‌.1‌ یادگیری ژرف در نگاه کلی 63](#_Toc475140646)

[4‌.2‌ پردازنده های گرافیکی NVidia: موتوری قوی برای یادگیری ژرف 67](#_Toc475140647)

[4‌.3‌ نرم افزار ها و toolkit های لازم برای یادگیری ژرف 69](#_Toc475140648)

[فصل چهارم جمع‌بندي و نتيجه‌گيري و پیشنهاداتجمع‌بندي و نتيجه‌گيري 71](#_Toc475140649)

[منابع و مراجع 73](#_Toc475140650)

|  |  |
| --- | --- |
| فهرست اشكال | صفحه |

[شکل 1: ترد، ترد بلاک و حافظهی سراسری در پردازنده‌های گرافیکی 9](#_Toc475136073)

[شکل 2: کرنل، گرید، بلاک و ترد در مدل میزبان - وسیله 10](#_Toc475136074)

[شکل 3: مدل میزبان، وسیله 11](#_Toc475136075)

[شکل 4: بانکهای حافظه در کودا 14](#_Toc475136076)

[شکل 5: معماری پردازندهی گرافیکی GeForce 8800 16](#_Toc475136077)

[شکل 6: TPC 19](#_Toc475136078)

[شکل 7: نمونهی تار پارچه 21](#_Toc475136079)

[شکل 8: اجرای SIMT 21](#_Toc475136080)

[شکل 9: معماری GeForce GT 200 25](#_Toc475136081)

[شکل 10: اجزای واحد TPC در پردازندههای گرافیکی سری 200 26](#_Toc475136082)

[شکل 11: معماری فرمی 31](#_Toc475136083)

[شکل 12: ساختار حافظه در فرمی 34](#_Toc475136084)

[شکل 13: فضای آدرس یکتا در معماری فرمی 36](#_Toc475136085)

[شکل 14: اجرای هم‌زمان کرنلها در فرمی 37](#_Toc475136086)

[شکل 15: پردازنده ی تسلا P100 43](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136087)

[شکل 16: بلوک دیاگرامی از P100 44](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136088)

[شکل 17: SMهای پردازنده ی تسلا P100 47](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136089)

[شکل 18: شتابدهنده p100 از جلو **Error! Bookmark not defined.**](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136090)

[شکل 19:شتابدهنده ی P100 از پشت 51](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136091)

[شکل 20:نمایشی از شمای دای و میکروبامپ ها 52](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136092)

[شکل 21: محل اینترپوزر 52](file:///C:\Users\Alireza\Downloads\Projects%20Template%20Amirkabir%20University%20(1).doc#_Toc475136093)

[شکل 22: NVLink 55](#_Toc475136094)

[شکل 23: هشت پردازنده گرافیکی که به صورت hyper cube قرار دارند 55](#_Toc475136095)

[شکل 24: 4 پردازنده گرافیکی که با استفاده از تکنولوژی NVLink به پردازنده اصلی و خود متصل اند. 56](#_Toc475136096)

[شکل 25: نمونه کد کودا 6 57](#_Toc475136097)

[شکل 26: یک سوپرکامپیوتر DGX-1 60](#_Toc475136098)

[شکل 27: یک پرسپترون که مدلی ساده از شبکه عصبی است 64](#_Toc475136099)

[شکل 28: یک مدل پیچیده از شبکه عصبی که نیازمند محاسبات سنگین است 66](#_Toc475136100)

[شکل 29: فریم وورک های مختلف برای کار با یادگیری ژرف 70](#_Toc475136101)

# فصل اول مقدمهمقدمه

پردازنده های گرافیکی امروزه در زمینه های گوناگونی از قبیل بازی ها، نرم افزار های گرافیکی همچون فوتوشاپ ، corel و ... مورد مصرف قرار میگیرند. با روی کار آمدن الگوریتم های شبکه ژرف و هوش مصنوعی در چندین سال اخیر و نیاز محاسباتی 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 است که برنامه های شبکه عصبی و یادگیری ژرف خود را در کمتر از دقیقه به اجرا در آورند، الگوریتم هایی که اجرای آنها بر روی سخت افزار های سنتی ممکن بود تا روز ها به طول انجامد. این شرکت با همکاری های وسیع با کمپانی های نرم افزاری و اینترنتی دنیا از قبیل فیسبوک و نتفلیکس و گوگل مسیر را برای بسیاری از الگوریتم های یادگیری ژرف هموار ساخته است. همچنین بسیاری از کمپانی های ماشین 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 شبیه­سازی 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 درصورتی‌که قرار باشد باکیفیت بالایی انجام شوند، از عهده­ی[[1]](#footnote-1)CPUها 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]](#footnote-2) را با کارت گرافیک[[3]](#footnote-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]](#footnote-4)آغاز می­کنیم و سپس بحث را با شرح معماری پردازنده­های گرافیکی ساخت شرکت NVIDIA ادامه خواهیم داد.

## کارت­های گرافیک

نخستین کارت گرافیک در سال 1960 میلادی توسط شرکت IBM ساخته شد. این کارت گرافیک که همراه رایانه­های IBM PC فروخته می­شد دارای 4 کیلوبایت حافظه بود و فقط قادر بود رنگ سبز را نشان دهد که از آن برای نمایش کارکترها استفاده می­شد. هرچند نخستین کارت گرافیک دارای قدرت پردازشی 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 افزایش قدرت آن­ها صنعت بازی­های رایانه­ای شروع به کارکرد. همچنین وجود کاربران متعدد باعث تضمین سود ناشی از فعالیت در بازار توسعه کارت­های گرافیکی شد. بدین­سان صنعت توسعه 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 گرفته‌شده‌اند. 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 می­شود 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[[5]](#footnote-5) اشاره کرد؛ این واسط می­تواند PCI[[6]](#footnote-6)، PCIe[[7]](#footnote-7)، [[8]](#footnote-8)AGP و ... باشد در حال حاضر[[9]](#footnote-9) بیشتر پردازنده­های گرافیکی از PCI e استفاده می­کنند. درگاه­های[[10]](#footnote-10) خروجی تصاویر نیز از دیگر ادوات جانبی هستند. این درگاه­ها که برای انتقال اطلاعات بر روی نمایشگر ساخته‌شده‌اند ممکن است دارای استاندارد DIV[[11]](#footnote-11) یا VGA[[12]](#footnote-12) 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 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[[13]](#footnote-13) ، 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 یک‌سو 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 استفاده از پردازنده­های گرافیکی برای کارهای عمومی و علمی بود. واقعیت این است که در حال حاضر 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[[14]](#footnote-14) و پردازش رأسی[[15]](#footnote-15) 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 برداشتن دو گام لازم بود. گام نخست تغییر معماری سخت­افزار و ارائه معماری مناسب پردازش 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[[16]](#footnote-16) به وجود آورد[[17]](#footnote-17).

## معماریCUDA[[18]](#footnote-18)

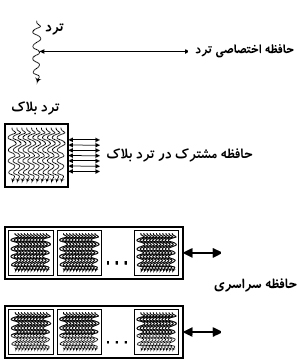
کودا­ که نخستین بار در سال 2006 هم‌زمان با ارائه­ی GeForce 8800 رونمایی شد، یک بستر نرم­افزاری[[19]](#footnote-19) برای توسعه­ی برنامه­های موازی بر روی برخی[[20]](#footnote-20) پردازنده­های گرافیکی ساخت شرکت NVIDIA را فراهم می­کند. با استفاده از این بستر نرم­افزاری، می­توان روی پردازنده­های گرافیکی دقیقاً مانند CPU برنامه­نویسی کرد. به این استفاده از پردازنده­های گرافیکی برای حل مسائل عمومی [[21]](#footnote-21)GPGPU گفته‌شده که گاهی اوقات به‌اختصار به‌صورت GP2U 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[[22]](#footnote-22).
4. پردازنده­ی گرافیکی دستورات را به‌صورت موازی بر روی‌داده‌هایی که در مرحله­ی یک دریافت کرده است اجرا می­کند.
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[[23]](#footnote-23) و پشتیبانی از عملیات منطقی بیتی.
* هم‌زمان‌سازی منطقی و آسان تردها
* امکان فراگیری آسان

یک برنامه­ی CUDA متشکل از یک سری کرنل[[24]](#footnote-24) 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 به‌صورت یک سری دسته که به آن­ها بلاک[[25]](#footnote-25) گفته می­شود تقسیم­بندی می­شوند. خود بلاک­ها هم در قالب یک سری گرید[[26]](#footnote-26) مرتب می­شوند. پس هر گرید شامل تعدادی بلاک و هر بلاک شامل تعدادی ترد است. 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 توضیح داده خواهد شد، هر ترد فقط بخشی 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[[27]](#footnote-27)، تعدادی رجیستر و حافظه­ی اختصاصی انحصاری است. در شکل ‏0‑1 این تقسیم­بندی­ها نشان داده‌شده است.

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

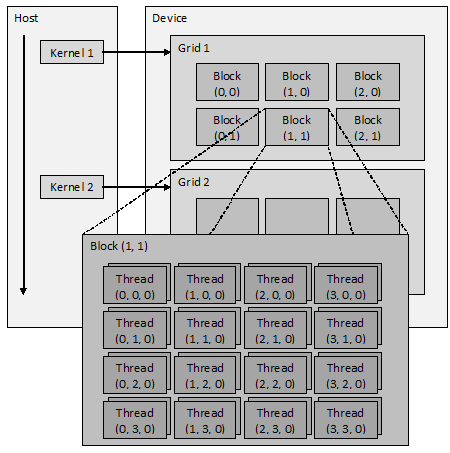
در معماری کودا ساختمان چینش ترد بلاک‌ها در داخل گرید می­تواند یک‌بعدی یا دوبعدی باشد. درعین‌حال ساختمان تردها هم در داخل ترد بلاک می­تواند یک، دو یا سه‌بعدی باشد. در فصول آتی در مورد ترد، ترد بلاک، گرید و کرنل بحث خواهیم کرد. در حال حاضر فقط کافی است با این مفاهیم یک آشنایی سطحی داشته باشید برای اینکه 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 گرفته‌شده است ترد می­گویند. يک Thread از تعدادي دستورالعمل تشکيل مي­شود که به ترتيب و بر روي يک هسته کودا اجرا مي­شود. علت انتخاب نام Thread (به معناي نخ) نيز به همين علت است. درواقع، Thread کوچک‌ترین جزء ساختار سلسله مراتبي مدل کودا است که اجراکننده‌ی دستورالعمل­هاي کرنل است. يعني درنهایت، يک کرنل توسط تعداد زيادي ترد اجرا مي­شود.

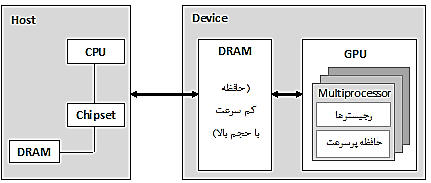
**ترد بلاک:** در داخل هر گرید تعدادی ترد بلاک (بلاک) وجود دارد. تردهایی که در داخل ترد بلاک هستند در حین اجرای برنامه می­توانند باهم داده رد و بدل نمایند.

**گرید:**در داخل هر کرنل فقط یک گرید وجود دارد. گرید یک تعریف است که نحوه­ی اختصاص منابع به کرنل را نشان می­دهد. به مجموعه­ای از ترد بلاک‌ها یک گرید می­گویند. در شکل ‏0‑2 رابطه­ی کرنل و گرید نشان داده‌شده است.

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

پیش­تر گفتیم که پردازنده­های قدیمی­تر دارای دو واحد مهم بودند که عبارت‌اند از واحد پردازش رأسی 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[[28]](#footnote-28) 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[[29]](#footnote-29) 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[[30]](#footnote-30)، وسیله[[31]](#footnote-31) استفاده می­شود. این مدل در شکل ‏0‑3 نشان داده‌شده است. همان­طور که در شکل نشان داده‌شده 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 ‏0‑3: مدل میزبان، وسیله

## حافظه در کودا

در يک پردازنده­ی گرافیکی منطبق بر کودا، شش نوع حافظه­ي مختلف با کاربردهاي متفاوت وجود دارد. این حافظه­ها را می­توان به دودسته‌ی حافظه­های روی تراشه و حافظه­های بیرون تراشه تقسیم کرد. حافظه­های بیرون تراشه ازنظر فيزيکي جزئي از یک DRAM محسوب شده و لذا ويژگي­هاي DRAM را دارند يعني داراي حجم بالا و سرعت پايين (درصورتی‌که cache نشوند، بين 200 الي 300 سيکل) مي­باشند. این حافظه­ها شامل موارد زیر هستند.

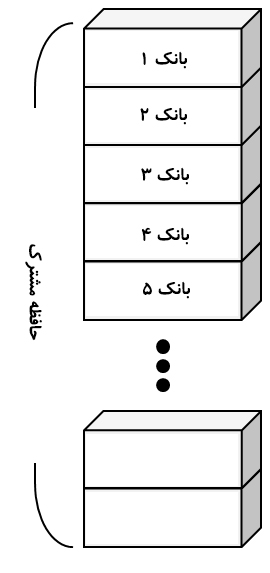
* **حافظه سراسری[[32]](#footnote-32):** هر پردازنده­ی گرافیکی يک نمونه از اين حافظه در اختیار دارد. این حافظه از سمت پردازنده­ی گرافیکی هم قابل‌خواندن و هم قابل‌نوشتن است. شایان‌ذکر است که این حافظه برخلاف حافظه­های ثوابت و بافت دارای حافظه­ی نهان نیست.
* **حافظه ثوابت[[33]](#footnote-33):** هر پردازنده­ی گرافیکی یک نمونه از این حافظه را در اختیار دارد. این حافظه از سمت پردازنده­ی گرافیکی فقط قابل‌خواندن است. حافظه­ی ثوابت همان‌طور که از نامش پيداست، حافظه­اي است براي ذخيره کردن مقادير ثابت. اين مقادير، مي­توانند آرايه­اي از اعداد باشند که قبلاً توسط پردازنده­ی اصلی به این حافظه منتقل‌شده و ترد آن‌ها را مي­خوانند. طول عمر اطلاعات موجود در حافظه ثوابت برابر با طول عمر برنامه است و در طول مدتي که کرنل­هاي مختلفي از يک برنامه اجرا مي­شوند، حافظه ثوابت مقادير خود را حفظ مي­کند. اين حافظه، دارای حافظه­ی نهان نیز هست. حافظه­ی نهان براي کاربردهايي که در آن­ها پردازنده­ی گرافیکی به‌صورت مکرر اعدادي را از حافظه­ي خود مي­خواند، استفاده مي­گردد. سرعت دسترسی به اطلاعات موجود در حافظه­ نهان برابر با سرعت دسترسی به حافظه مشترک است پس به‌صورت کلی سرعت اين حافظه می­تواند بين 1 تا 300 سيکل ساعت تغییر کند. بايد توجه داشت که استفاده­ي اشتباه و يا بیش‌ازحد نياز از این حافظه تنها احتمال عدم اصابت[[34]](#footnote-34) را زياد مي­کند.
* **حافظه بافت[[35]](#footnote-35):** هر پردازنده­ی گرافیکی دارای یک حافظه­ی بافت با دسترسی خواندن و نوشتن است. کلمه­ي بافت در اصطلاح گرافيکي به معناي تصويري است که بر روي سطح اشياي سه­بعدي نمايش داده مي­شود. در کودا اين حافظه نيز کاربردهاي منحصر به خود را دارد. اين حافظه نیز داراي حافظه­ی نهان است. با توجه به اينکه این حافظه، جزئي از DRAM به‌حساب مي­آيد، سرعت دسترسی به اطلاعاتی از آن‌که در حافظه­ی نهان موجود نیست بسیار پایین است. اين حافظه براي حالتي بهینه‌سازی شده است که بخواهیم آرايه­هاي دوبعدي ذخيره­ کنیم و تردهاي بخواهند درايه­هايي نزديک به هم را از آن بخوانند. برای حافظه بافت، منطق ذخیره‌سازی اطلاعات در حافظه­ی نهان مطابق با نيازهاي پردازش گرافيکي بافت­ها است. علاوه بر اين، حجم حافظه بافت قابل تنظيم است و حداکثر مي­تواند برابر با کل فضاي DRAM باشد. اما استفاده­ي بیش‌ازحد از آن، تنها باعث کاهش نرخ اصابت[[36]](#footnote-36) و سرعت خواندن از آن مي­شود.
* **حافظه محلی[[37]](#footnote-37):** هر ترد بخشي از اين حافظه را، به‌صورت انحصاري، براي خواندن و نوشتن در اختيار دارد. هرگاه اطلاعات یک ترد در رجيسترها جا نشود و نخواهيم این اطلاعات را در حافظه­ی مشترک قرار دهیم باید این اطلاعات را به حافظه­ی محلی سپرد. حافظه­هاي محلی بر روي DRAM قرار دارند پس دارای حجم بسیار بزرگ‌تری نسبت به حافظه­ی مشترک می­باشند. هر ترد به بخشي از حافظه­ي DRAM تحت عنوان حافظه­ی محلی دسترسي 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 حافظه­هاي روي تراشه هستند. اين حافظه­ها داراي سرعتي بالاتر (يک سيکل) و حجمي پايين­تر (حافظه­ی مشترک[[38]](#footnote-38) معمولاً 16 یا 48 کيلوبايت بوده و ثبات­ها معمولاً 16384 يا 8192 عدد هستند[[39]](#footnote-39)) هستند.

* **حافظه­ مشترک:** همه­ی تردهای موجود در یک بلاک، به‌صورت اشتراکي از يک نمونه از اين حافظه براي خواندن و نوشتن استفاده مي­کند.
* **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 کيلوبايت بوده و از 16 بانک يک کيلوبايتي با کلمات 32 بيتي تشکيل مي­شود. هر بانک مي­تواند در هر سيکل به يک آدرس سرويس­دهي کند. به‌عبارتی‌دیگر پهناي باند هر بانک، 32 بيت در هر سيکل ساعت است. بنابراين تعداد سرويس­هاي هم‌زمانی که حافظه مشترک مي­تواند بدهد، به تعداد بانک­هايش است.

شکل ‏0‑4 نحوه­ي تقسيم يک حافظه­ی مشترک را به بانک­هاي ذکرشده، نمايش مي­دهد. کلمات 32 بيتي متوالي، به بانک­هاي متوالي تخصيص داده مي­شوند.



شکل ‏0‑4: بانک­های حافظه در کودا

اگر درآن‌واحد، چندين ترد تقاضاي دسترسي به يک بانک را داشته باشند، تداخل پيش مي­آيد و پردازنده­ی گرافیکی ناچار است که به تقاضاها، به‌صورت نوبتي پاسخ دهد. در صورت بروز تداخل بانک، سرعت نهايي حافظه مشترک افت مي­کند. تقسيم حافظه مشترک به بانک­هاي متعدد، احتمال تداخل در دسترسي به بانک­ها را کاهش داده و سرعت نهايي خواندن و نوشتن در آن را افزايش مي­دهد.

در حالت عادي، تداخلي بين Threadهاي دونیمه‌ی Warp وجود ندارد؛ زيرا Thread هاي نيمه­ي دوم Warp صبر مي­کنند تا Thread هاي نيمه­ي اول کارشان با Shared Memory تمام شود. هرچند اين اتفاق مي­تواند نوعي تداخل بانک محسوب شود، اما نظر به اينکه هر Warp شامل 32 Thread بوده و تنها 16 بانک مستقل براي پاسخ‌دهی به نيازهاي اين Thread ها وجود دارد، بنابراين تداخل بين دونیمه‌ی يک Warp امری است اجتناب‌ناپذیر و معمولاً از تأخیر به وجود آمده صرف‌نظر مي­شود.

با توجه به این‌که آدرس­هاي يک Shared Memory معمولاً 32 بيتي هستند، در حالتي که با آرايه­اي سروکار داريم که اعضاي آن‌ها متغيرهاي غير 32 بيتي (مثل char يا int64) هستند، بايستي مراقب تداخل در بانک­ها باشيم.

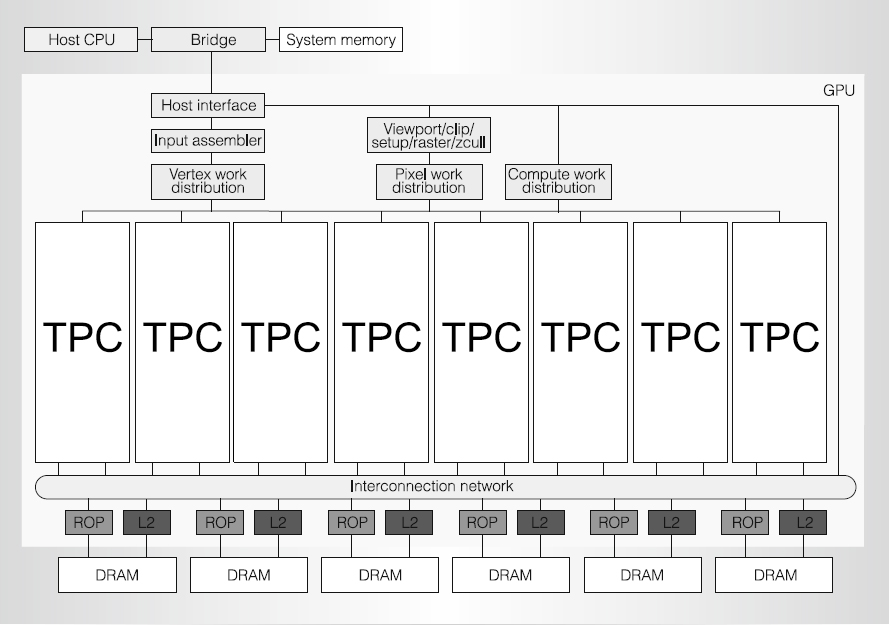
دو شکل­ 4-7، حالتي را معرفي مي­کنند که هيچ تداخلي در دسترسي Thread ها به بانک­هاي Shared Memory وجود ندارد. شکل سمت چپ آدرس­دهي خطي با گام يک، و شکل سمت راست، آدرس‌دهی تصادفي یک‌به‌یک را نشان مي­دهد.

## 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[[40]](#footnote-40)، پیکسلی[[41]](#footnote-41)، 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.
* **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 ‏0‑5 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 واردشده، پردازش می­شود و نهایتاً از قسمت پایین شکل، اطلاعات پردازش‌شده خارج می­شود. در ادامه واحدهای مختلف این پردازنده شرح داده‌شده است.



شکل ‏0‑5: معماری پردازنده­ی گرافیکی GeForce 8800

1. آرایه پردازنده­های جریان ([[42]](#footnote-42)SPA)

واحد SPA که در شکل نشان داده نشده است، شامل مجموعه­ی تمام واحدهای TPC[[43]](#footnote-43) است.این واحد مسئولیت اجرای برنامه­های گرافیکی، اجرای برنامه­های پردازش موازی، کنترل و مدیریت تردها را بر عهده دارد.

1. رابط میزبان[[44]](#footnote-44)

در ادبیات پردازش موازی با پردازنده­های گرافیکی، به پردازنده­ی اصلی میزبان و به پردازنده­ی گرافیکی وسیله[[45]](#footnote-45) گفته می­شود. بنابراین، واحد رابط میزبان همان­طور که از اسمش پیداست وظیفه­ی هماهنگی با میزبان را بر عهده دارد. این واحد باید دستورات CPU را دریافت کرده، اطلاعات موردنیاز را از حافظه فرابخواند و سازگار بودن دستورات مختلف را بررسی کند.

1. 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.

1. 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.

1. واحدهای توزیع کار

واحدهای توزیع کار رأسی[[46]](#footnote-46)، پیکسلی[[47]](#footnote-47) و پردازشی[[48]](#footnote-48) مسئولیت توزیع پردازش­های مختلف را در بین واحد­های TPC بر عهده دارند. وظیفه­ی این واحدها توزیع بار یکنواخت بین TPCهاست.

1. واحد ROC[[49]](#footnote-49)

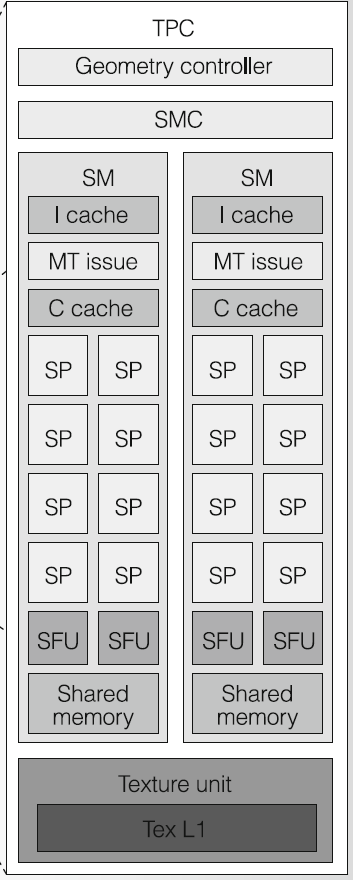
هر واحد ROC به یک قسمت مشخص از حافظه دسترسی دارد. ­TPC اطلاعات را از طریق شبکه­ی اتصالات داخلی[[50]](#footnote-50) برای ROC ارسال می­کند.وظیفه­ی ROC کنترل عمق رنگ، تست رنگ، مخلوط کردن و به‌روزرسانی رنگ­هاست. واضح است که تمامی این واحدها به‌صورت موازی باهم کار می­کنند.

1. 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[[51]](#footnote-51) و حافظه­ی سراسری[[52]](#footnote-52) بر روی این 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.

1. واحد TPC

همان­طور که در شکل ‏0‑6 نشان داده‌شده است، هر واحد TPC دارای یک کنترل‌کننده‌ی اطلاعات هندسی[[53]](#footnote-53)، یک واحد کنترل‌کننده‌ی [[54]](#footnote-54)SM، یک واحد بافت[[55]](#footnote-55) و دو SM[[56]](#footnote-56) 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 نوشته‌شده 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.



شکل ‏0‑6: TPC

مهم­ترین بخش TPC واحد SM است. این واحد دارای یک کش دستورالعمل[[57]](#footnote-57)، یک کش فقط خواندنی ثوابت[[58]](#footnote-58)، یک واحد واکشی و اجرای دستورالعمل[[59]](#footnote-59)، 16 کیلوبایت حافظه­ی مشترک خواندنی و نوشتنی، هشت پردازنده جریان[[60]](#footnote-60)، و دو واحد پردازش­های ویژه[[61]](#footnote-61) است.هرکدام از SPها دارای یک واحد MAD[[62]](#footnote-62) است که عملیات ضرب و جمع متوالی به فرم را در یک پالس ساعت انجام خواهد داد.­SPها یا پردازنده­های­ جریان به‌تدریج با نام هسته­های کودا شناخته شدند. پس‌ازاین ما نیز برای خطاب کردن پردازنده­های جریان (SP) لفظ هسته­های کودا را به کار خواهیم برد.

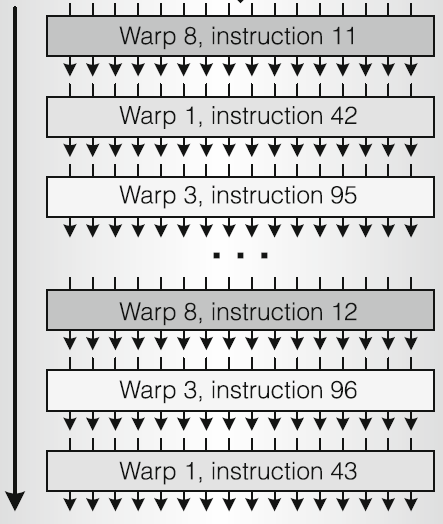
در هر SM واحد SFU دو وظیفه­ی عمده بر عهده دارد. وظیفه­ی نخست اجرای عملیات ریاضی سطح بالا مانند عملیات محاسبه­ی توابع مثلثاتی و جذر بوده و وظیفه­ی دوم الحاق پیکسل­ها برای نمایش است. آخرین قسمت یک SM حافظه­ی مشترک است. حافظه­ی مشترک به برنامه­هایی که بر روی هسته­های کودای موجود در یک SM اجرا می­شوند این امکان را می­دهد که بدون نیاز به دسترسی به حافظه­ی خارجی به تبادل اطلاعات بپردازند. بدین‌سان ترافیک دسترسی به حافظه­ی خارجی کاهش‌یافته و سرعت نهایی اجرای برنامه افزایش خواهد یافت.در GeForce 8800 فرکانس پالس ساعت هسته­های کودا و واحدهای SFU برابر با GHz 1.5 است و قدرت محاسباتی هر SM در بیشترین حالت برابر با GFLOPS 36 است. می­توان گفت کوچک‌ترین واحد پردازنده­ای 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 می­شود. 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[[63]](#footnote-63) 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[[64]](#footnote-64) 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[[65]](#footnote-65) 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 [[66]](#footnote-66)SIMT استفاده می­شود. واحد مدیریت اجرای تردهای SM، تردها را در دسته­های 32تایی که به آن وارپ[[67]](#footnote-67) گفته می­شود اجرا می­کند. واژه­ی وارپ هم مثل ترد از صنعت نساجی به عاریه گرفته‌شده است و به معنی تار پارچه است. تار پارچه مجموعه­ای از یک سری نخ موازی با هم است و اگر به شکل ‏0‑7 توجه کنید متوجه خواهید شد که علت این نام‌گذاری چیست.

|  |
| --- |
|  |
| شکل ‏0‑7: نمونه­ی تار پارچه |
|  |

هر SM قادر است 24 وارپ را به‌صورت هم‌زمان اجرا کند. تمامی تردهایی که در یک وارپ قرار دارند مشابه هم هستند و باید از یک آدرس برنامه شروع به اجرا کنند اما آن­ها آزادند که در طول اجرای برنامه به قسمت­های مختلفی جهش کرده و به صورت مستقل اجرا شوند؛ هرچند که اگر چنین اتفاقی رخ دهد اجرای آن­ها به صورت موازی نخواهد بود. به این سبک اجرای موازی تردها SIMT گفته می­شود. در هر SM واحد اجرای SIMT بررسی می­کند و یکی از 24 وارپی را که برای اجرا آماده است انتخاب می­کند.اولین انتخاب بر اساس نوع وارپ، نوع دستورالعمل 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 ‏0‑8 نشان داده‌شده است.



شکل ‏0‑8: اجرای SIMT

پس‌ازاینکه یک وارپ برای اجرا انتخاب شد SM، تردهای آن را بر روی هسته­های کودا به‌صورت مستقل از هم اجرا می­کند. بهترین بازده، زمانی به دست می­آید که برنامه­ای که قرار است اجرا گردد دارای انشعاب وابسته به داده­ نباشد. درصورتی‌که چنین پرشی وجود داشته باشد اصطلاحاً می­گویند واگرایی اتفاق افتاده است. هر زمان که در یک SM واگرایی اتفاق بیفتد تردهای آن به‌صورت متوالی اجرا خواهند شد. اگر بخواهیم دو روش SIMT و SIMD را باهم مقایسه 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 سطح بالا برنامه­های گرافیکی و برنامه­های پردازش موازی را مستقیماً 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 بهینه­سازی به دستورالعمل­های قابل‌اجرا توسط پردازنده­ی گرافیکی تبدیل می­شوند. بدین ترتیب شرکت می­تواند نسل­های متفاوتی از پردازنده­های گوناگون تولید کند ولی زبان برنامه­نویسی را تغییر ندهد. لازم به ذکر است که تغییر مداوم زبان برنامه­نویسی، کاربران را نسبت به استفاده 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 دستورها شامل جمع، ضرب، ضرب-جمع یا همان MAD، یافتن کمینه، بیشینه، مقایسه و دستورهای تبدیل بین اعداد صحیح و اعشاری هستند. همچنین یک سری دستورات دیگر هم هست که توسط واحد SFU اجرا می­شود. این دستورات عبارت‌اند از محاسبه­ی سینوس، کسینوس، مقدار نمایی، لگاریتم باینری، معکوس کردن عدد و محاسبه­ی جذر.
  + این عملیات قادرند پرچم­های[[68]](#footnote-68) صفر، منفی، کری[[69]](#footnote-69) و سرریز[[70]](#footnote-70) را که به‌صورت اختصاصی در هر ترد قرار دارند برای استفاده­های بعدی ست کنند.
* 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[[71]](#footnote-71) 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 است نهایتاً تا هشت CTA را به‌صورت هم‌زمان اجرا کند. این تعداد وابسته است به منابعی که هر CTA نیاز دارد. واحد SMC درصورتی‌که 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 می­شود. 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 برنامه‌نویسی کرد، CTAها کاملاً مستقل از هم اجرا می­شوند 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 به‌صورت موازی باهم اجرا شوند و بر روی پردازنده­ی گرافیکی دیگر هم به‌صورت متوالی اجرا شوند در این حالت تنها تفاوتی که حاصل می­شود در زمان اجرای برنامه است و اشتباهی 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.

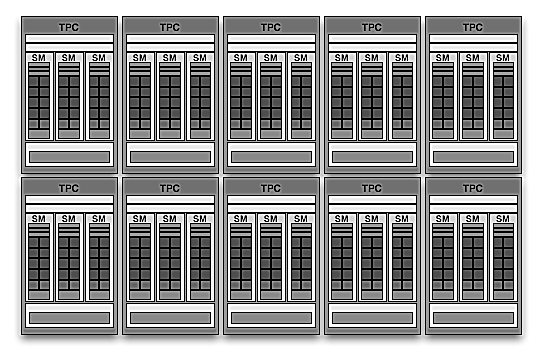
با این تفاسیر تطبیق مدل برنامه­نویسی کودا با معماری سخت­افزاری کار دشواری نیست. هر ترد در مدل برنامه‌نویسی کودا با یک ترد در سخت­افزار معادل است. هر بلاک کودا معادل CTA و گرید کودا با گریدی که در این قسمت معرفی شد برابر است. پس بلاک­هایی که در کودا تعریف می­شود در قالب CTA در SM اجرا خواهد شد.

## معماری پردازنده های گرافیکی GeForce سری 200

پردازنده­های گرافیکی سری 200 دومین نسل پردازنده­های گرافیکی شرکت NVIDIA با معماری یکپارچه واحد سایه زنی هستند. شرکت NVIDIA پردازنده­های گرافیکی سری 200 را با هدف دستیابی به اهداف زیر تولید کرد.

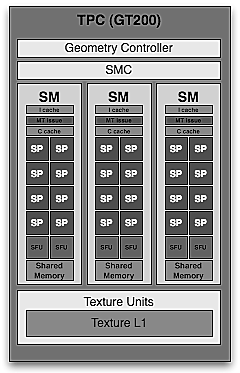
* بالا بردن کارایی پردازنده­های گرافیکی نسل قبل یعنی GeForce سری 8
* بهینه­سازی معماری برای بازی­های رایانه­ای جدید
* بهینه­سازی معماری برای کاهش مصرف انرژی و فضای مصرفی
* ساخت یک پردازنده­ی گرافیکی قدرتمند برای اجرای پردازش موازی با مقیاس بزرگ­تر

پردازنده­ی گرافیکی GeForce GTX 200 اولین پردازنده­ای است که از نسل دوم واحد سایه زنی یکپارچه شرکت NVIDIA استفاده می­کند. این پردازنده دارای سرعتی به‌صورت 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 استفاده‌شده است درحالی‌که در CPUها به‌صورت 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 ‏0‑9: معماری GeForce GT 200

در شکل ‏0‑9 معماری GeForce GT 200 نشان داده‌شده است. در بالاترین قسمت واحد برنامه­ریز ترد[[72]](#footnote-72) وجود دارد که اجرای تردهای مختلف بین TPCها را مدیریت می­کند. در قسمت پایین هم واحد­های بافت و واحد دستورالعمل­های اتمیک[[73]](#footnote-73) 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 ‏0‑10 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 ‏0‑10: 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 می­شد. مرحله­ی نخست پس از محاسبه­ی حاصل‌ضرب بود و مرحله­ی دوم پس از محاسبه­ی حاصل جمع. این گرد کردن دومرحله‌ای باعث افت دقت می­شد. در پردازنده­های سری 200 عملیات MAD روی اعداد اعشاری با دقت مضاعف، حاصل را پس از عملیات ضرب گرد نکرده و آن را با حفظ تمامی رقم­های اعشاری با مقدار ثابت نهایی جمع خواهد کرد. به این روش محاسبه که نسبت به روش قبلی از دقت بالاتری برخوردار است Fused Multiply ADD گفته می­شود.

پردازنده­های GeForce سری 200 در بسیاری از جزئیات ازجمله واحدهای بافت، محاسبات هندسی، واحدهای ROC، مدیریت حافظه و مدیریت مصرف انرژی نسبت به نسل­های پیش از خود برتری داشتند که بیان به‌تفصیل این جزئیات از موضوع بحث این کتاب خارج است. پردازنده­های گرافیکی GeForce سری 200 با توانایی­های بالایی که داشتند 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[[74]](#footnote-74) ارائه کرد.

## معماری Fermi

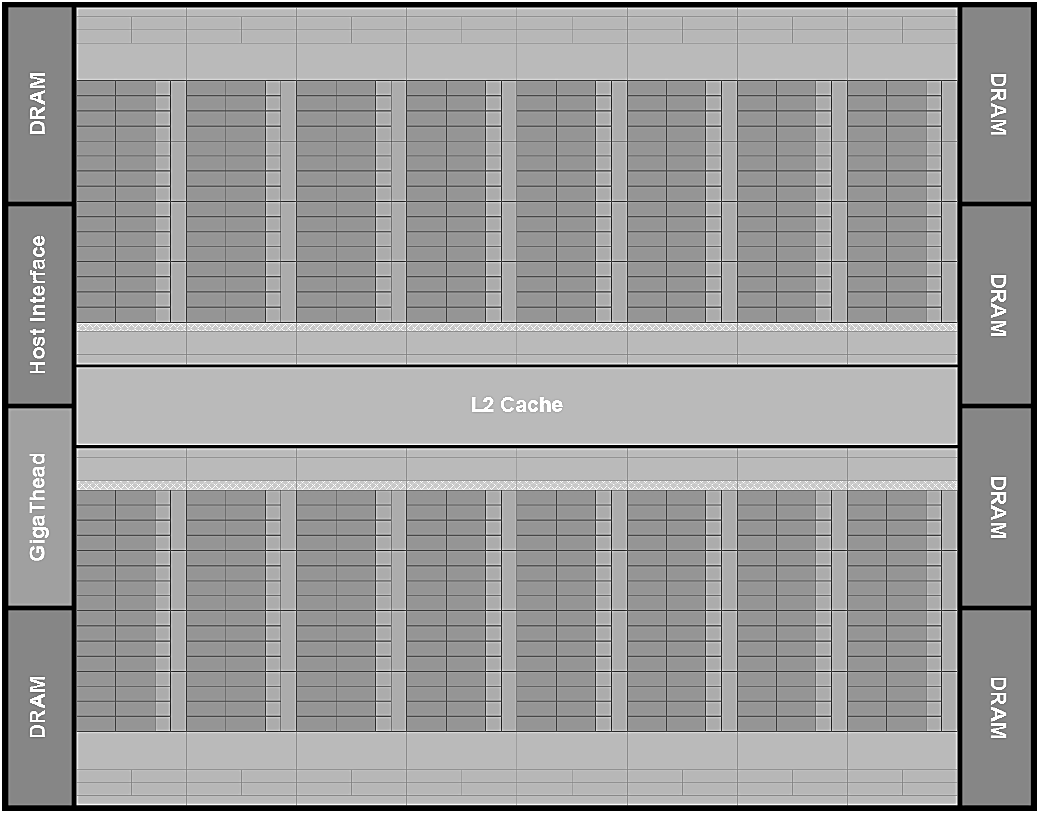
زمانی که شرکت NVIDIA شروع به توسعه­ی معماری فرمی کرد، پردازش موازی با کمک پردازنده­های گرافیکی دیگر امری نوپا نبود. در آن زمان برنامه­های بسیاری برای دو نسل قبلی از پردازنده­های گرافیکی توسعه پیداکرده بود و نظرات بسیاری از سوی کاربران برای شرکت ارسال‌شده بود. بدین ترتیب NVIDA توانست با بهره­گیری از تجربیات 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 شده است در ادامه­ ذکرشده است.

* بهینه­سازی کارایی واحد محاسبات اعشاری با دقت مضاعف[[75]](#footnote-75)
  + هرچند واحد محاسبات اعشاری با دقت معمولی در پردازنده­های گرافیکی قبلی دارای کارایی قابل قبولی بود اما برخی از مسائل مربوط به پردازش­های موازی نیازمند کارایی بالا در کار با اعداد اعشاری با دقت مضاعف هستند.
* استقرار واحد تشخیص و تصحیح خطای [[76]](#footnote-76)ECC
  + 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، محاسبات حساس 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 گرفته‌شده است استفاده کنند به همین دلیل امکان استفاده از کش واقعی نیز در معماری فرمی در نظر گرفته‌شده است.
* افزایش حجم حافظه مشترک[[77]](#footnote-77)
  + 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 است که بارها کار خود را تغییر دهند. این تغییر کار[[78]](#footnote-78) فرآیندی زمان­بر است 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 داشت. در معماری فرمی این دستورات نیز گنجانده‌شده است. دستورالعمل­های اتمیک به دستورالعمل­هایی گفته می­شود که اجازه­ی انجام عملیات بازیابی، پردازش و ذخیره مجدد قسمت­های مختلفی از یک ساختمان داده را به‌صورت هم­زمان به تردهای مختلف می­دهد. عملیاتی اتمیک نامیده می­شوند که شامل دستورهای بازیابی، پردازش و ذخیره برای هر ترد به‌صورت مستقل و بدون ایجاد مزاحمت با سایر تردها هستند. 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 مناسب و متناسب است. 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 محاسبات 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 دستورالعمل­ها را به‌صورت مستقل بین دو وارپ مختلف به‌صورت هم‌زمان توزیع و اجرا نماید. بالاخره اینکه در معماری فرمی 64 کیلوبایت حافظه­ی رم اضافه شد که به‌صورت پویا قابل تقسیم به دو قسمت 16 و 48 کیلوبایتی بین کش L1 و حافظه­ی مشترک بود.
* نسل دوم دستورالعمل­های موازی اجرای ترد
  + در این نسل فضای آدرس را برای تطبیق­پذیری کامل با C++ بهینه کرده­اند و همچنین واحد محاسبات اعشاری با دقت ساده و مضاعف ایجاد شد که کاملاً با استاندارد IEEE 754 سال 2008 میلادی مطابقت دارد. همچنین در این نسل واحد پیش­بینی انشعاب[[79]](#footnote-79) ایجادشده است که تأثیر به سزایی در افرایش کارایی پردازنده­های گرافیکی دارد.
* بهینه­سازی واحدهای حافظه
  + 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 ‏0‑11 نمونه­ای از یک پردازنده­ی گرافیکی با معماری فرمی را نشان می­دهد.



شکل ‏0‑11: معماری فرمی

همان­طور که شکل ‏0‑11 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.

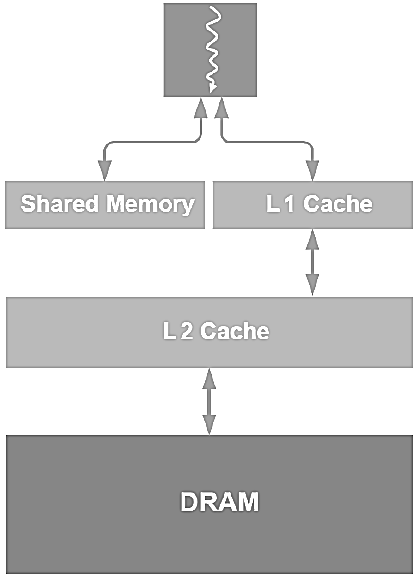
### SMهای نسل سوم

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[[80]](#footnote-80) و یک واحد محاسبات اعشاری[[81]](#footnote-81) است که هردوی آن­ها دارای خط لوله­ کامل هستند. در نسل سوم SMها از استاندارد IEEE 754 سال 2008 به‌صورت کامل پشتیبانی می­شود. این پردازنده­ی گرافیکی دارای یک واحد FMA[[82]](#footnote-82) 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 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 محاسبه شود. از طرف دیگر هر SM دارای 4 واحد SFU[[83]](#footnote-83) بوده که قادرند دستورات 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 محاسبات اعشاری مشغول است امکان استفاده از سایر واحدهای SM وجود دارد. محاسبات اعشاری 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 می­سازد تا بتواند به‌صورت 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 دستورالعمل صحیح، دو دستورالعمل اعشاری، یک دستورالعمل صحیح و یک دستورالعمل اعشاری، دستورهای ذخیره و بازیابی و دستورهای واحد SFU می­توانند به‌صورت هم‌زمان اجرا شوند.

### حافظه­ با قابلیت بازپیکربندی

حافظه­ی مشترک نقش مهمی را در برنامه­هایی که با پردازنده­ی گرافیکی نوشته می­شوند 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 داشتند. این مشکل در پردازنده­های فرمی حل شد. در فرمی هر SM دارای 64 کیلوبایت حافظه­ است 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 8 فضایی که برای کش در نظر گرفته‌شده 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 به‌صورت 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 ‏0‑12: ساختار حافظه در فرمی

### واحد تصحیح و تشخیص خطا

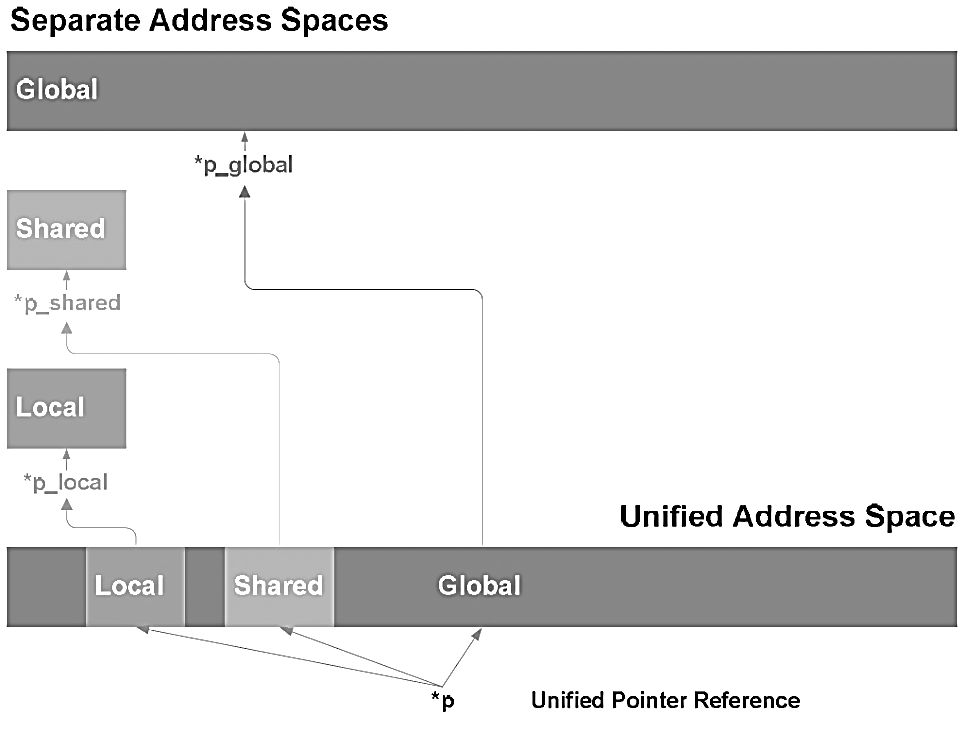
درصورتی‌که قرار باشد یک پردازنده در مسائل حساس مورداستفاده قرار بگیرد، باید حتماً صحت ثبت و انتقال اطلاعات روی باس­های مدارهای آن بررسی شود. فرمی اولین پردازنده­ی گرافیکی است که می­تواند این بررسی را انجام دهد. فرمی که مجهز به واحد ECC[[84]](#footnote-84) است که قادر است خطای­هایی را که منجر به تغییر 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 متوقف‌شده 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 L2آن 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 بروز خطا در قسمت­های مختلف، پردازنده­ی فرمی را برای استفاده شدن در سیستم­های پردازش موازی که نسبت به خطا حساس هستند میسر می­سازد.

### تکنولوژی PTX 2.0[[85]](#footnote-85)

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 نوشته‌شده‌اند توسط درایور پردازنده­ی گرافیکی به زبان ماشین تبدیل می­شوند. 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 ایجادشده است که از طریق آن می­توان به تمام قسمت­های حافظه دست پیدا کرد.

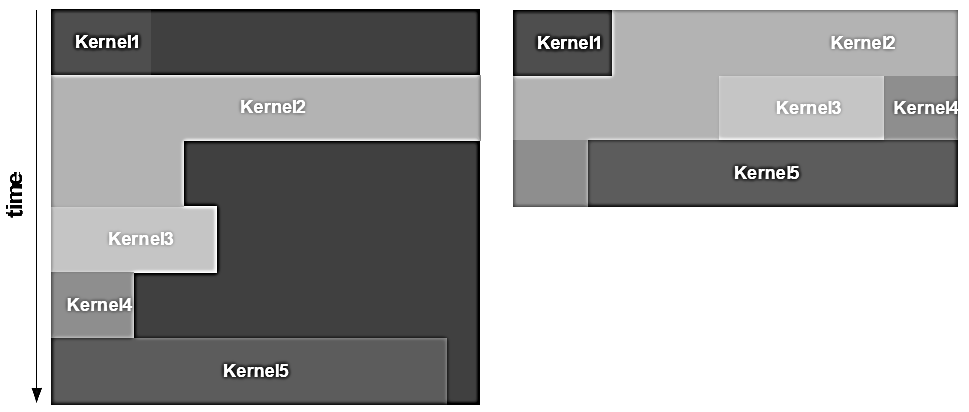


شکل ‏0‑13: فضای آدرس یکتا در معماری فرمی

در معماری فرمی قابلیت­های کار با اعداد اعشاری هم بهبود پیدا کرد. در فرمی اعداد غیرمعمولی[[86]](#footnote-86) به‌صورت پیش­فرض توسط سخت­افزار قابل‌شناسایی شد. همین­طور چهار روشی که در سند IEEE 754 2008 برای گرد کردن اعداد در نظر گرفته‌شده 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 هستند که بین صفر و کمترین عدد غیر صفر قرار دارند. در نسل­های 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.

### 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 استفاده از تمام قدرت پردازنده­ی گرافیکی فراهم شده و سرعت نهایی اجرای برنامه بیشتر خواهد شد. این موضوع در شکل ‏0‑14 نشان داده‌شده است.



شکل ‏0‑14: اجرای هم‌زمان کرنل­ها در فرمی

### کاستی­ها فرمی

علیرغم کارایی بالای فرمی هنوز هم کاستی­هایی 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 فوق‌سریع مطرح می­گردد این است که برنامه و تمامی اطلاعات موردنیاز آن، همه با هم در حافظه­ی پردازنده هستند. بر اساس قانون آمدال که در زمان توسعه­ی mainframeها بیان شد در یک سیستم پردازش فوق‌سریع لازم است 1 بایت حافظه و 1 بایت بر ثانیه ورودی و خروجی به ازای هر دستورالعمل بر ثانیه وجود داشته باشد پس در سامانه­ای با توانایی پردازش GFLOPS 500 باید GB 500 حافظه وجود داشته باشد. هرچند ممکن است بسیاری تصور کنند که این حجم حافظه بیش‌ازحد بزرگ است اما نباید این نکته را فراموش کرد که توسعه­ی پردازنده­های فوق‌سریع فقط برای حل مسائل دشوارتر است. مسائلی که به دلیل وجود ابعاد وسیع هنوز برای بشر قابل‌حل نیستند.

1. عدم توانایی اتصال مستقیم ورودی و خروجی به پردازنده­ی گرافیکی

بسیاری از برنامه­های واقعی نیاز دارند که حجم بسیار زیادی از داده­ها را به‌عنوان ورودی دریافت کرده و همین­طور حجم وسیعی از داده­ها را به‌عنوان خروجی بازگردانند. در حال حاضر اطلاعات باید از ورودی‌ها و خروجی­ها گرفته‌شده و در اختیار CPU قرار گیرند و سپس 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 است که با کمک پردازنده­های گرافیکی کار می­کنند. درصورتی‌که بتوان معماری را به‌گونه‌ای تغییر داد که پردازنده­های گرافیکی بتواند به‌صورت مستقیم[[87]](#footnote-87) به ورودی­ها و خروجی­ها و یا به دیسک سخت دسترسی پیدا کند، سرعت و کارایی پردازنده­های گرافیکی بهتر خواهد شد.

1. عدم پشتیبانی از حافظه­های سلسله مراتبی

با توجه به اینکه کاربران مختلف نیاز به سیستم­هایی با ابعاد گوناگون دارند، معمولاً سازندگان تراشه­های حافظه، آن­ها را طوری طراحی می­کنند که امکان قرارگیری دو، چهار و یا حتی هشت 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 شود که بتواند از چنین سلسله مراتبی پشتیبانی کند. در درجه­ی دوم باید این ترتیب قرارگیری حافظه­ها ازلحاظ نرم­افزاری هم موردبررسی قرار گیرد و هماهنگی­های لازم برای امکان کار با چنین تراشه­هایی فراهم شود. درصورتی‌که پردازنده­های گرافیکی بتوانند از این سلسله‌مراتب حافظه پشتیبانی کنند، مشکل نخست نیز حل خواهد شد زیرا توسعه‌دهندگان سامانه­ها قادر خواهند بود با توجه به نیاز سیستم برای پردازنده­ی گرافیکی آن، حافظه در نظر بگیرند.

مقایسه­ی نسل­های مختلف پردازنده­های گرافیکی

برای اینکه جمع­بندی آنچه تاکنون بیان شد آسان­تر گردد در جدول ‏0‑1 جزئیات سه نسل اصلی پردازنده­های گرافیکی شرکت NVIDIA را با هم مقایسه می­کنیم.

جدول ‏0‑1: پردازنده­های گرافیکی نسل­های مختلف شرکت NVIDIA

|  |  |  |  |
| --- | --- | --- | --- |
| GPU | G80 | GT200 | Fermi |
| تعداد ترانزیستور (میلیون) | 681 | 1400 | 3000 |
| هسته­های کودا | 128 | 240 | 512 |
| توانایی انجام محاسبات اعشاری با دقت مضاعف | ندارد | FMA ops/clock 30 | FMA ops/clock 256 |
| 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 | KB 16 | KB 16 | قابل انتخاب بین KB 48 یا KB 16 |
| کش L1 | ندارد | ندارد | قابل انتخاب بین KB 48 یا KB 16 |
| کش L2 | ندارد | ندارد | KB 768 |
| واحد ECC | ندارد | ندارد | دارد |
| تعداد کرنل­ها هم‌زمان | 1 | 1 | 16 |
| پهنای باند حافظه | 32 بیت | 32 بیت | 64 بیت |

### NVIDIA TESLA C2050

پردازنده­های تسلای سری 20 که بر اساس معماری فرمی طراحی‌شده‌اند 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 ‏0‑2: ویژگی­های پردازنده­ی تسلا C2050 و C2070

|  |  |
| --- | --- |
| ویژگی | توضیح |
| 448 هسته­ی کودا | توان ارائه­ی GFLOPS 515 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[[88]](#footnote-88) | 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. |

# فصل سوم معماری پاسکالمعماری 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 محاسبات با حجم های بالا بر روی بستر های مختلف دیتاسنتر 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.



## 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 استفاده از چندین پردازنده گرافیکی به منظور محاسبات سریع.

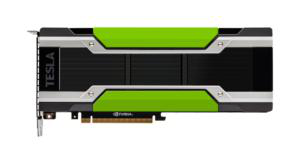
NVLink – مدل جدید شرکت Nvidia به منظور ارتباط سریع با پهنای باند بالا بین چنین کارت پردازنده گرافیکی. این تکنولوژی قادر است بیشترین میزان پردازنده گرافیکی را به هم متصل نماید.

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



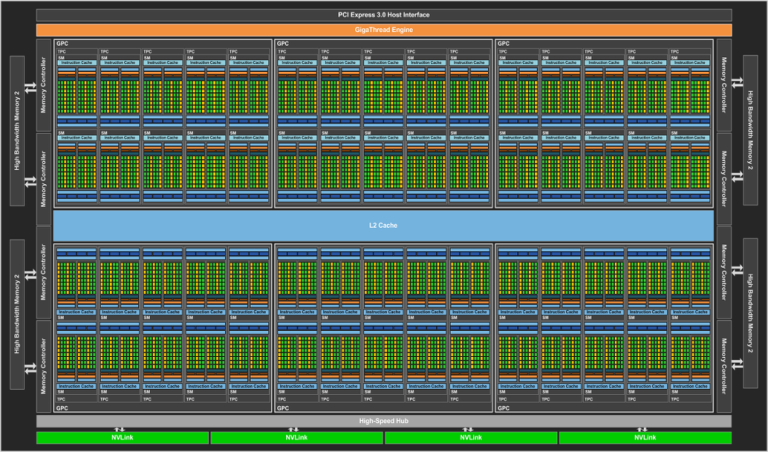
## Tesla P100 پیشنهادی عالی برای تکنولوژی NVLink و سرورهای PCI-Express

پردازنده Tesla P100 به دو صورت میتوانند با هم ارتباط برقرار کنند. یک راه سنتی که پردازنده های گرافیکی با استفاده از پورت های PCI-Express به هم و به پردازنده های گرافیکی خارج از سرور متصل خواهد شد. راه دیگیری از استفاده از ماژول SXM2 برای سرور های NVLink میباشد. پردازنده ی P100 برای سرور های مبتنی بر بستر PCI-express این امکان را برای دیتاسنتر ها فراهم میکند که تمامی پردازند های اصلی 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 صورت توزیع شده بر روی چند پردازنده گرافیکی به منظور محاسباتی سنگین همچون یادگیری سریع فراهم آورد

## 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 اقزایش 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 SM ها 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 سازی عملکرد پردازند گرافیکی در محاسبات و همچنین کاهش میزان انرژی به نسبت با این عملکرد دو هدف اصلی در این معماری پردازنده گرافیکی است. میزان تغییراتی که در SMهای این معماری نسبت 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) 4096-bit HBM2 (16GB) |
| **Memory Bandwidth** | 288 GB/s | 288 GB/s | 732 GB/s | 549 GB/s (12GB) 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 mm² | 601 mm² | 610 mm² | 610 mm² |
| **Manufacturing Process** | 28-nm | 28-nm | 16-nm | 16-nm |

## Pascal SM

هر واحد SM از پردازنده GP100 شامل 64 واحد single precision (FP32) است. 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 مدریتی وارپ ها و دو واحد توزیع می‌باشند. این درحالی است که پردازنده GP100 نصف تعداد پردازنده های کودا را نسبت 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 وحود 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 پاسکال یک مدل مدیریتی برای افزایش سرعت عملیات های Load و store فراهم کرده که به مراتب عملکرد این دستور العمل ها را برای واحد های ممیز شناور افزایش داده است. سیستم مدریت SM ها در معماری پاسکال یک سیستم هوشمند بوده به طوری که درمقایسه با معماری مکسول عملکردی بهتر را نشان میدهد.

## 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 است که این نسبت در مقایسه با پردازنده Kepler GK110 3 به 1 است. 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:

رشد روز افزون رشته هایی از قبیل یادگیری ژرف این خواسته را به وجود آورده است که یک وسیله محاسباتی 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 محاسباتی تکینیکی که نیازمند عملگرد سریع واحدهای اعشاری است، 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 عصبی مصرف میکند، که این امر این امکان را به ما میدهد تا شبکه عصبی بزرگ تری هنگام آموزش شبکه و یا اجرای شبکه طراحی کنیم. استفاده از FP16 عملکرد شبکه عصبی را تا دو برابر نسبت به استفاده از عملیات FP32 بالا برده است و به طور مشابه استفاده 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.

## 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 صورت موازی به اجرا در آمده اند عملیات های خواندن- تغییر- نوشتن 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 مناسبی 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 بخشی 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 است.در مدل های قبل برای جمع های FP64 از حلقه های CAS استفاده میشد که به طور کلی بسیار کند تر مدل های سخت افزاری و محلی است.

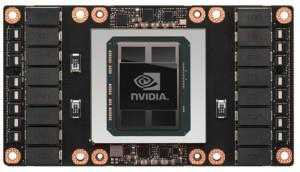
قابلیت محاسباتی 6.0

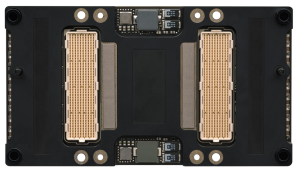
GP100 قابلیت محاسباتی جدید 6.0 را پشتیبانی میکند. جدول زیر تفاوتهای پارامترهای مختلف قابلیت های محاسباتی را نشان می دهد.

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |  |

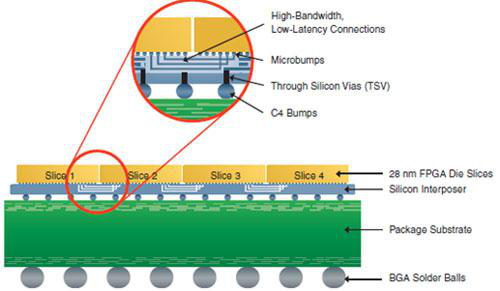
## بهبودی شگرف در حافظه

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 پردازشی 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

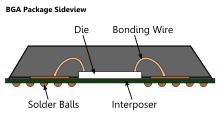
تسلا P100 هر دوی این محدودیت ها را با استفاده از تکنولوژی حافظه های پشته ای برطرف کرده است. تکنولوژی ای که چندین لایه از واحد های DRAM را قادر میسازد تا به صورت عمودی در یک پکیج درون پردازنده گرافیکی با هم در ارتباط باشند. تسلا P100 اولین پردازنده گرافیکی است که با استفاده از تکنولوژی پهنای باند سریع 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.

بر خلاف GDDR5 که شامل 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[[89]](#footnote-89) و microbump ها مرتبط هستند.

شکل 18:شتابدهنده ی P100 از پشت

هر دای 8 گیگابیت HBM2، شامل 5000 سوراخ های درون-سیلیکان می باشد. در نهایت یک اینترپوز[[90]](#footnote-90) 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.

شکل 19:نمایشی از شمای دای و میکروبامپ ها

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=imgres&cd=&cad=rja&uact=8&ved=0ahUKEwjWpZqU_YXSAhXE6xQKHerjAEkQjRwIBw&url=https://en.wikipedia.org/wiki/Interposer&psig=AFQjCNGoeYOyy7sVAEfuievKNyL5-tUSRg&ust=1486831514620507)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 است که این پهنای باند در حالت پیک میباشد) به طوری که این پهنای باند سه برابر بیشتر از مدل تسلا M40 است.

شکل 20: محل اینترپوزر

## ECC Memory

از دیگر مزیت های تکنولوژی HBM2 میتوان به پشتیبانی محلی از رفع خطاهای کد[[91]](#footnote-91) 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 معناست 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[[92]](#footnote-92) و شناسای خطای دوتایی[[93]](#footnote-93) (SECDED) پشتیبانی میکند.

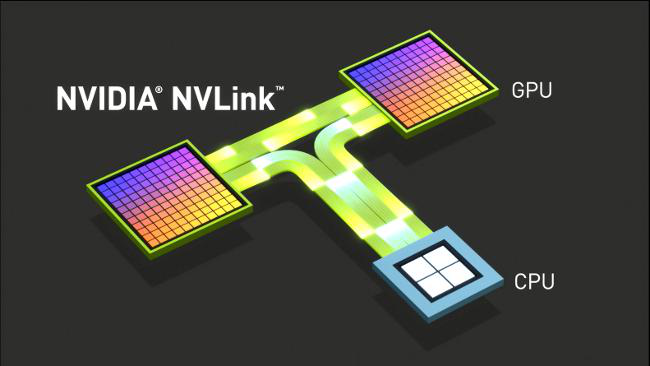
## NVLink برای ارتباط سریع

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.

امروزه، استفاده از چند پردازنده گرافیکی در یک ایستگاه‌های پردازشی[[94]](#footnote-94) 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.

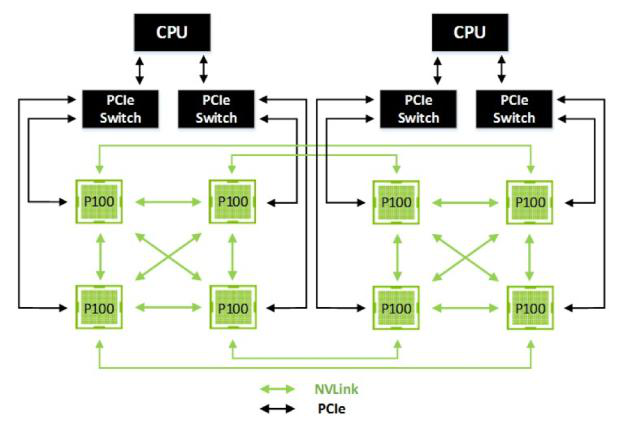
پشتیبانی از GPU ISA بدین معنا است که بتوان با استفاده از تکنولوژی NVLink پردازنده های گرافیکی را به هم متصل کند و برنامه ما به طور مستقیم بر روی داده هایی که در چند پردازنده گرافیکی قرار دارد اجرا شود. پردازنده گرافیکی همچنین میتواند عملیات اتومیک حافظه ای را بر روی پردازنده های گرافیکی کنترلی از راه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

NVLink از ارتباطات سریع سیگانلی انویدیا[[95]](#footnote-95) 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[[96]](#footnote-96) 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.



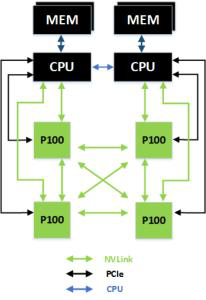
شکل 21: NVLink

تصویر زیر 8 پردازنده گرافیکی به صورت مش های مکعبی ترکیبی[[97]](#footnote-97) 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

در حالی که وظیفه اصلی NVLink وصل کردن چندین پردازنده گرافیکی GP100 به یکدیگر است، این امکان را که پردازنده گرافیکی GP100 را به پردازنده های اصلی IBM متصل نمود، وجود دارد. تصویر زیر معرف این موضوع است که چهار پردازنده گرافیکی با استفاده از تکنولوژی NVLink به CPUها متصل شده است. در این پیکربندی، هر پردازنده گرافیکی مجموعا دارای سرعت 120 گیگابیت بر ثانیه به صورت دو طرفه با 3 پردازنده گرافیکی دیگر است و 40 گیگابیت ارتباط دوطرفه با پردازنده اصلی است.



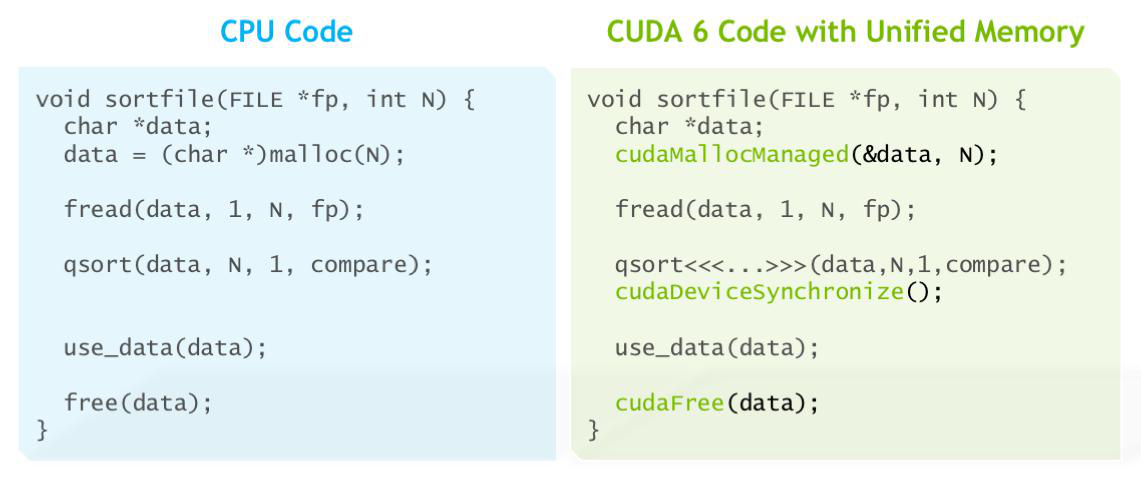
شکل 23: 4 پردازنده گرافیکی که با استفاده از تکنولوژی NVLink به پردازنده اصلی و خود متصل اند.

## افزایش بهره وری برنامه نویسی با استفاده از حافظه یکپارچه[[98]](#footnote-98)

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

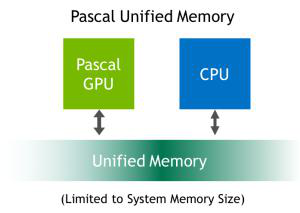


شکل 24: نمونه کد کودا 6

### حافظه یکپارچه پردازنده P100

با توجه به مزیت هایی که حافظه یکپارچه کودا 6 برای کاربر به ارمقان آورد، در مدل پاسکال 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 پوشاندن حافظه های اصلی نسل 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.



پشتیبانی پردازنده گرافیکی از خطایابی آدرس دهی صفحه یکی از بنیادی ترین تغییراتی است که در این نسل 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 است. اول از همه بایستی گفت که عیبیابی صفحه بدین معناست که پلتفرم کودا دیگر نیازی به همزمان سازی سیستم 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 می‌شود که صفحه مورد نظر به صورت خودکار به درون حافظه پردازنده گرافیکی انتقال یابد. متناوبا، ممکن است که صفحه مورد نظر به درون حافظه پردازنده گرافیکی نگاشته شود 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[[99]](#footnote-99) سرعت عمل بالاتری نسبت به انتقال داده درون پردازنده گرافیکی دارد.

فلذا با توجه به این سیستم خطایابی صفحه در نسل جدید پردازنده های گرافیکی و کودا 8، وابستگی داده های global و حافظه ی یکپارچه تضمین خواهد شد. این بدان معناست که با پردازنده GP100، میتوان به صورت همزمان پردازنده های گرافیکی و پردازنده های اصلی به حافظه یکپارچه دسترسی داشته باشند. گفتنی است که این امر در مدل های کپلر و مکسول قابل قبول نبود. به این دلیل که هیچ تضمینی وجود نداشت، 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 است. ( می توانید به عکس زیر مراجعه کنید. در این سیستم، استفاده از حافظه یکپارچه به صورت پیشفرض مورد استفاد هاست. با این حساب دیگر نیازی به یک تخصیص دهنده خاص برای استفاده تخصیص حافظه نخواهد بود.



این نکته نیز نبایست فراموش شود که چندین دستورعمل خاص به منظور فعال سازی این متد بر روی سیستم عامل نیازمند است که شرکت NVidia با همکاری Red Hat و انجمن های لینوکس این عملگر قوی را فعال کرده اند.

## NVidia DGX-1 یادگیری ژرف با استفاده از سوپرکامیپوترها

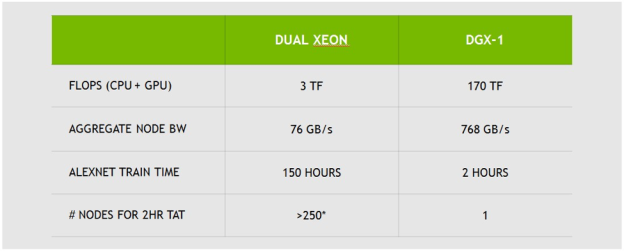
دانشمندانی که در حوزه پردازش اطلاعات و یا هوش مصنوعی کار میکنند، به دقت بالا، سادگی و در عین حال سرعت بالا برای سیستم های یادگیری ژرف نیازمندند. آموزش هر چه سریع تر باشد، نمو خلاقیت های پیرو این حیطه از علم و همچنین فروش محصولات وابسته به آن در بازار جهانی سریع تر خواهد بود. پردازشگر NVidia DGX-1 اولین سیستمی است که به منظور پردازش های ژرف طراحی شده است، به طوری که یک پک کامل از سخت افزار ها و نرم افزار های لازم به منظور اجرای سریع و آسان 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 پردازشگر اصلی متصل خواهد بود. گفتنی است که در بدترین حالت، استفاده از سرور NVidia DGX-1 در حالتی که از 8 پردازشگر P100 استفاده کند، دوازده برابر افزایش سرعت در حالتی از نسخته قبلی پردازشگر ها استفاده شود، دارد.



# 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 اصلی از علم هایی نظیر یادگیری ژرف و هوش مصنوعی بر آورده کردن نیاز هایی از قبیل ساخت یک ماشین که بتواند به صورت هوشمندانه به اسان خدمت کند. این هوشمندی بدین صورت است که نیازی به یک دستور عمل خاص و یا الگوریتم های مشخص نداشته و ماشین به صورت خودکار با یادگیری از پیرامون خود نیازهای بشری را ارضا نماید. یادگیری ژرف یکی از مهمترین دست 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، تصمیماتی درست و به جا در موقعیت های خاص بگیرد. اما ازعان به این نکته ضروری است که آموزش به چنین ماشینی نیاز به حجم عظیمی از داده دارد. به علاوه، در مدل های یادگیری ژرف که در آن ها از لایه های زیادی استفاده شده است، پردازش این اطلاعات را برای کاربر سخت و طاقت فرسا 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 هسته 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 سخت افزار را یکجا برای انجام چنین پردازشی فراهم آورد. در همین هنگام، شرکت NVidia با همکاری با دانشگاه Stanford تصمیم به استفاده از پردازنده های گرافیکی برای پردازش های مورد نیاز الگوریتم های یادگیری ژرف گرفتند. نتیجه ی چنین تصمیمی، چیزی نشد جز آن که 12 پردازنده گرافیکی از شرکت NVidia توانایی یادگیری ژرف همگام با 2000 پردازنده اصلی را برآورد کرد. همانطور که از اعداد مشخص است این کاهش میزان پردازنده، نه تنها در میزان مصرف انرژی صرفه جویی بسزایی کرد، بلکه این امکان را به دیگر شرکت ها داد تا با تهیه تعدادی که پردازنده گرافیکی کمتر از تعداد انگشتان دست، به رقابت با پردازنده های Intel در الگوریتم های یادگیری ژرف بپردازند.

خیلی ها بر این باورند که انقلاب یادگیری ژرف به سال 2012 که در این سال مسابقه 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 میشود را راه اندازی کردند و نتیجه چیزی نشد جز آن که عملکردی دوچندان بهتر نسبت به رقبایی که از کلاستر های قوی برای این مسابقه استفاده کرده بودند، به دست آورد. لازم به ذکر است که کریژوسکی و تیمش هیچ کدی در زمینه پردازش تصویر ننوشتند. بلکه متد آنها بدین صورت بود که با استفاده از یادگیری ژرف این امکان را با ماشین دادند تا خودش شروع به یادگیری کند. آنها با طراحی یک شبکه عصبی موسوم به AlexNet و آموزش آن با استفاده از میلیون ها مثال که خود نیازمند تریلیون ها محاسبات 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.

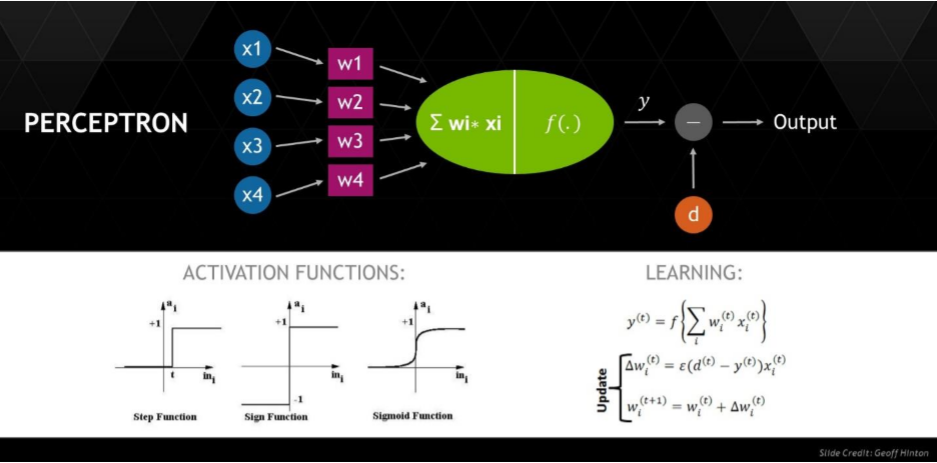
پس از آن، رفته رفته، شبکه های ژرف عصبی بر روی پردازنده های گرافیکی انقلابی را در زمینه های پردازش تصویر به طور خاص و به طور کلی الگوریتم های وابسته به یادگیری راه انداختند. کاربرد این علم به مراتب فراتر از درک ماست، به عنوان مثال استفاده از ماشین هایی که بدون سرنشین قادر به رانندگی بر روی هر سطحی و تحت هر شرایطی هستند و یا مترجم هایی که بلادرنگ هر زبانی هر با زبان دلخواه ما ترجمه سلیس میکند. امروزه، استفاده از یادگیری ماشین در کنار پردازنده های گرافیکی تمام امکانات فوق الذکر را برای ما فراهم کرده است. به طوری که استفاده از پردازنده های گرافیکی به طور چشمگیری مرحله زمان آموزش شبکه عصبی را کاهش داده است.

## یادگیری ژرف در نگاه کلی

یادگیری ژرف یک تکنیکی است که مغز اسنان را با استفاده از شبکه ها عصبی و متدهای یادگیری عصبی شبیه سازی می کند. با یادگیری بیشتر، ماشین 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 اشکال هندسی مختلفی را بدون نیاز به والدین خود شناسایی کند. به طور مشابه، یادگیری ژرف و یا یادگیری عصبی این امکان را به سیستم میدهد که اشیا را از تصاویر شناسایی و آن ها را دسته 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 مشخص شده است. 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[[100]](#footnote-100) چندین ورودی که هر کدوم مشخص کننده یک ویژگی از داده ورودی است، وجود دارد. وظیفه نهایی این سلسله پرستپرون ها آموختن و شناسایی 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 پرسپترون داده شده است یک وزن به خصوص که میزان اهمیت آن ویژگی را مشخص میکند نشان میدهد.



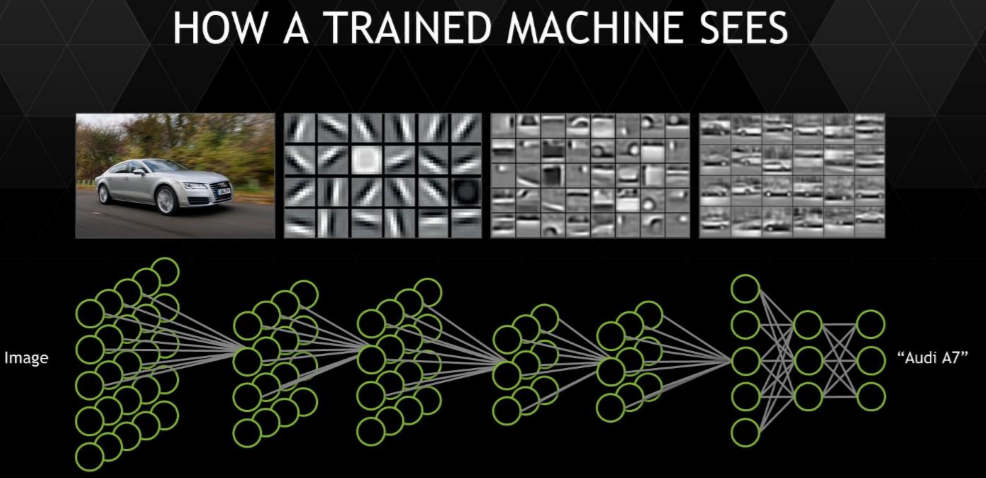
شکل 26: یک پرسپترون که مدلی ساده از شبکه عصبی است

برای مثال، یک پرسپترون را در نظر بگیرید که هدف آن شناسایی شماره صفر در عکس های دست نوشته است. واظهاً، عدد صفر به صورت های گوناگونی میتواند نوشته شود که این صور به دست 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 تقسیم میکند. به عنوان مثال این عکس میتواند به 4 بخش که مشخص کننده 4 ویژگی است تقسیم می‌شود. منحنی قسمت راست بالایی به ویژگی شماره 1 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[[101]](#footnote-101) 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 شناسایی دقیق اعداد باشد. این پروسه از خطای ناشی از جواب فیدبک گرفته و بنا به این فیدبک ماتریس را دستخوش تغییر داده. این روش را نشر بازپس خطا[[102]](#footnote-102) 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 شناسایی و طبقه بندی درست اشیا را انجام داد، برنامه حاضر 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[[103]](#footnote-103) میتوان به شناسایی اعداد در دستخط ها برای استفاده از چک های بانگی در دستگاه های خودپرداز و یا شناسایی عکس های دوستان در سامانه امنیتی فیسبوک، پیشنهاد فیلم بر اساس 50 میلیون کاربر شبکه اینترنتی نتفلیکس[[104]](#footnote-104)، شناسایی 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 متصل هستند، تشکیل شده باشد؛ هر کدام از این گره ها مشخص 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 شکل بالا، اولین لایه، از شبکه عصبی، ویژگی های اصلی از عکس را به قسمت های مختلف میشکند تا الگوهای اولیه از تصویر مانند خط ها و زووایا به دست 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 میسازد و در لایه بعدی مدل خودرو را مشخص میکند و در نهایت در لایه آخر این شبکه عصبی برند خودرو را مشخص میکند. که در این مثال خودرو آوودی[[105]](#footnote-105) A7 بود.

یک راه دیگر به جای استفاده از شبکه کامل وصل[[106]](#footnote-106) استفاده از لایه های کانولوشنی[[107]](#footnote-107) 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[[108]](#footnote-108) 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[[109]](#footnote-109) گویند.

## 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.

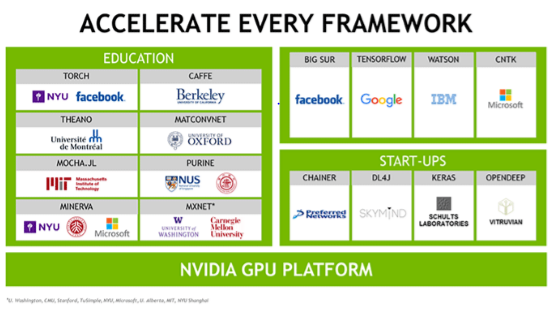
امروزه دیگر این امر مشخص است که یادگیری این شبکه ها چه در زمینه های صنعتی و چه در زمینه های آموزشی با استفاده از پردازنده های گرافیکی صورت میپذیرد، چرا که این پردازنده ها نسبت به مدل های همتای خود یعنی پردازنده های اصلی (CPU) هم از لحاظ مصرف انرژی و هم از لحاظ سرعت اجرای برنامه ها به صورت موازی بسیار 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.

## 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[[110]](#footnote-110)، گوگل، IBM، فیسبوک، بایدو[[111]](#footnote-111) 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 موسستات تحقیقاتی. شرکت NVidia به منظور هر مصرفی از خانگی گرفته تا سیستم های سوپرکامیپوتر 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.



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

# فصل چهارم جمع‌بندي و نتيجه‌گيري و پیشنهاداتجمع‌بندي و نتيجه‌گيري

به طور کلی میتوان گفته که پردازنده های گرافیکی شرکت NVidia با رشد فزاینده ای که این سال ها در زمینه های گوناگون داشته اند، به بزرگترین پرچم دار سخت 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 نسل جدید از معماری پردازنده گرافیکی تحت عنوان پاسکال در را برای دیگر پردازش های موازی نیز بازتر کرد و این امکان را به کاربران داده است که کار با اعداد اعشاری را با سرعتی دو برابر نسل 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 است، نسل جدیدی از حافظه های یکپارچه میباشد که کاربر را از تخصیص دهی حافظه به صورت تخصصی رهایی میبخشد و عملا این امکان را به حافظه اصلی و حافظه گرافیکی میدهد که با هم و در یک زمان به یک نقطه از حافظه دسترسی داشته باشند. گفتنی است 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 رضایت برنامه نویسان نیازمند پلتفرمی کاربردوستدارانه تر است، چرا که هر برنامه‌ای که به زبان کودا نوشته می‌شود تنها مختص به معماری ای است که برنامه نویس برای آن نوشته است، میباشد. فلذا این پلتفرم باید این امکان را برای کاربران و برنامه نویسان فراهم کند که فارغ از معماری فعال در سیستم، 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 است که بر روی معماری های اصلی اجرا می‌شوند. این پردازنده هر چند به منظور کارهای گرافیکی طراحی شده است 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 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Central Processing Unit [↑](#footnote-ref-1)
2. Graphics Processing Unit [↑](#footnote-ref-2)
3. Graphic Card [↑](#footnote-ref-3)
4. Compute Unified Device Architecture [↑](#footnote-ref-4)
5. Mainboard [↑](#footnote-ref-5)
6. Peripheral Component Interconnect [↑](#footnote-ref-6)
7. PCI Express [↑](#footnote-ref-7)
8. Accelerated Graphic Port [↑](#footnote-ref-8)
9. سال 2013 [↑](#footnote-ref-9)
10. Port [↑](#footnote-ref-10)
11. Digital Video Interface [↑](#footnote-ref-11)
12. Video Graphics Array [↑](#footnote-ref-12)
13. Rendering pipeline [↑](#footnote-ref-13)
14. Pixel Fragment Processor [↑](#footnote-ref-14)
15. Vertex Processor [↑](#footnote-ref-15)
16. Compute Unified Device Architecture [↑](#footnote-ref-16)
17. 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. [↑](#footnote-ref-17)
18. برای راحتی از اینجا به بعد واژه­ی CUDA را به صورت «کودا» خواهیم نوشت. [↑](#footnote-ref-18)
19. Platform [↑](#footnote-ref-19)
20. 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. [↑](#footnote-ref-20)
21. General-Purpose computing on Graphics Processing Units [↑](#footnote-ref-21)
22. Run kernel [↑](#footnote-ref-22)
23. نمونه­های جدید پردازنده­های گرافیکی منطبق با کودا از اعداد اعشاری نیز به خوبی پشتیبانی می­کنند. [↑](#footnote-ref-23)
24. Kernel [↑](#footnote-ref-24)
25. Block [↑](#footnote-ref-25)
26. Grid [↑](#footnote-ref-26)
27. Program Counter [↑](#footnote-ref-27)
28. Vertex Shading [↑](#footnote-ref-28)
29. Pixel Shading [↑](#footnote-ref-29)
30. Host [↑](#footnote-ref-30)
31. Device [↑](#footnote-ref-31)
32. Global memory [↑](#footnote-ref-32)
33. Constant memory [↑](#footnote-ref-33)
34. Cache miss [↑](#footnote-ref-34)
35. Texture memory [↑](#footnote-ref-35)
36. Cache hit [↑](#footnote-ref-36)
37. Local memory [↑](#footnote-ref-37)
38. Shared Memory [↑](#footnote-ref-38)
39. واضح است که این اعداد در آینده تغییر خواهند کرد. [↑](#footnote-ref-39)
40. Vertex process [↑](#footnote-ref-40)
41. Pixel process [↑](#footnote-ref-41)
42. Streaming Processor Array [↑](#footnote-ref-42)
43. Texture/Processor Cluster; Thread Processing Cluster [↑](#footnote-ref-43)
44. Host Interface [↑](#footnote-ref-44)
45. Device [↑](#footnote-ref-45)
46. Vertex Work Distribution [↑](#footnote-ref-46)
47. Pixel Work Distribution [↑](#footnote-ref-47)
48. Compute Work Distribution [↑](#footnote-ref-48)
49. Raster Operation Processor [↑](#footnote-ref-49)
50. Interconnection Network [↑](#footnote-ref-50)
51. Local Memory [↑](#footnote-ref-51)
52. Global Memory [↑](#footnote-ref-52)
53. Geometry Controller [↑](#footnote-ref-53)
54. SM Controller (SMC) [↑](#footnote-ref-54)
55. Texture Unit [↑](#footnote-ref-55)
56. Streaming Multiprocessor [↑](#footnote-ref-56)
57. Instruction Cache (I cache) [↑](#footnote-ref-57)
58. Constant Cache (C cache) [↑](#footnote-ref-58)
59. Multithread Instruction Fetch And Issue (MT issue) [↑](#footnote-ref-59)
60. Streaming Processor [↑](#footnote-ref-60)
61. Special Function Unit [↑](#footnote-ref-61)
62. Multiply ADD [↑](#footnote-ref-62)
63. Not A Number [↑](#footnote-ref-63)
64. Sub normal [↑](#footnote-ref-64)
65. Overhead [↑](#footnote-ref-65)
66. Single Instruction Multiple Thread [↑](#footnote-ref-66)
67. Warp [↑](#footnote-ref-67)
68. Flag [↑](#footnote-ref-68)
69. Carry [↑](#footnote-ref-69)
70. Overflow [↑](#footnote-ref-70)
71. Cooperative Thread Array [↑](#footnote-ref-71)
72. Thread Scheduler [↑](#footnote-ref-72)
73. Atomic [↑](#footnote-ref-73)
74. Fermi [↑](#footnote-ref-74)
75. Double Precision Floating Point [↑](#footnote-ref-75)
76. Error Correcting Code [↑](#footnote-ref-76)
77. Shared Memory [↑](#footnote-ref-77)
78. Task Switching [↑](#footnote-ref-78)
79. Branch Prediction [↑](#footnote-ref-79)
80. ALU [↑](#footnote-ref-80)
81. FPU [↑](#footnote-ref-81)
82. Fused Multiply-Add [↑](#footnote-ref-82)
83. Special Function Unit [↑](#footnote-ref-83)
84. Error Correcting Code [↑](#footnote-ref-84)
85. Parallel Thread eXecution (PTX) 2.0 [↑](#footnote-ref-85)
86. Sub normal [↑](#footnote-ref-86)
87. از طریق DMA [↑](#footnote-ref-87)
88. Asynchronous Data Transfer [↑](#footnote-ref-88)
89. Through-silicon [↑](#footnote-ref-89)
90. Interposer [↑](#footnote-ref-90)
91. Error correct code (ECC) [↑](#footnote-ref-91)
92. Single error correct [↑](#footnote-ref-92)
93. Double error correct [↑](#footnote-ref-93)
94. Workstation [↑](#footnote-ref-94)
95. Nvidia’s High-speed Signaling interconnect (NVHS) [↑](#footnote-ref-95)
96. gangs [↑](#footnote-ref-96)
97. Hybrid cube mesh [↑](#footnote-ref-97)
98. Unified memory [↑](#footnote-ref-98)
99. mapping [↑](#footnote-ref-99)
100. Perceptron [↑](#footnote-ref-100)
101. Forward propagation [↑](#footnote-ref-101)
102. Backward propagation [↑](#footnote-ref-102)
103. Deep neural network [↑](#footnote-ref-103)
104. Netflix [↑](#footnote-ref-104)
105. Audi [↑](#footnote-ref-105)
106. Fully connected [↑](#footnote-ref-106)
107. Convolutional layer [↑](#footnote-ref-107)
108. Retinal ganglion [↑](#footnote-ref-108)
109. Convolutional Neural Networks (CNNs) [↑](#footnote-ref-109)
110. Amazon [↑](#footnote-ref-110)
111. Baidu [↑](#footnote-ref-111)