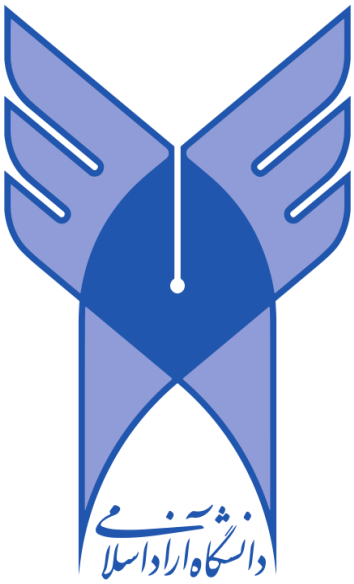

موضوع پروژه: پردازنده‌های گرافیکی (GPU) و کاربرد آن‌ها در پردازش‌های موازی

نام و نام خانوادگی اعضای گروه: علی ابوالحسن مطلق - محمد اسعدیان

شماره دانشجویی: ۴۰۲۷۲۱۷۷۶ - ۴۰۲۷۱۵۶۰۲

نام درس و سکشن آن: معماری کامپیوتر ، یکشنبه ها از ۱۰:۱۵ تا ۱۳:۱۵

نام استاد: استاد جواد حسن لی

تاریخ: ۱۴۰۴/۰۲/۲۶

مقدمه

در دهه‌های اخیر، رشد شتابان علوم رایانه و ظهور مسائل پیچیده محاسباتی در حوزه‌هایی همچون هوش مصنوعی، تحلیل کلان‌داده‌ها، شبیه‌سازی علمی، و پردازش تصویر، نیاز به سامانه‌های پردازشی سریع‌تر و کارآمدتر را بیش از پیش برجسته کرده است. پردازنده‌های مرکزی (CPU) که به‌طور سنتی مسئول اجرای دستورالعمل‌های سیستم‌های رایانه‌ای بوده‌اند، در پاسخ‌گویی به این نیازها با محدودیت‌های ساختاری مواجه شده‌اند. CPUها در مواجهه با وظایفی که قابلیت تقسیم و اجرای هم‌زمان دارند (مانند ماتریس‌برداری، پردازش هم‌زمان تصویر یا داده‌های حسگرها)، بازدهی شان به‌صورت قابل‌توجهی کاهش می‌یابد.

از سوی دیگر، ماهیت بسیاری از مسائل نوین محاسباتی، به‌گونه‌ای است که می‌توان آن‌ها را به واحدهای کوچکتر تقسیم کرد و به‌صورت موازی مورد پردازش قرار داد. برای مثال، در پردازش ویدئوهای HD در زمان واقعی، هزاران یا میلیون‌ها عملیات عددی تکراری باید به‌صورت هم‌زمان و سریع انجام شوند. در چنین شرایطی، پردازنده‌های گرافیکی (GPU) با برخورداری از تعداد بالای هسته‌های پردازشی، توانایی اجرای صدها تا هزاران نخ (thread) را به‌صورت هم‌زمان دارند. با این وجود، بهره‌برداری بهینه از این توان نیازمند درک صحیح از معماری GPU، اصول برنامه‌نویسی موازی و آگاهی از محدودیت‌ها و چالش‌های موجود است.

مسئله اصلی این تحقیق آن است که علی‌رغم پیشرفت چشم‌گیر در توسعه و استفاده از GPUها، هنوز بسیاری از پژوهشگران، توسعه‌دهندگان نرم‌افزار، و حتی متخصصان حوزه رایانش، آگاهی دقیقی از ساختار GPU و قابلیت‌های آن در پردازش‌های موازی ندارند. همچنین، در سطح زیرساخت‌های بومی، استفاده از GPUها غالباً محدود به حوزه گرافیک یا بازی‌سازی بوده و در سایر زمینه‌ها توسعه نیافته است. در نتیجه، شناسایی ظرفیت‌ها، کاربردها، مزایا و موانع استفاده از پردازنده‌های گرافیکی در پردازش‌های موازی، به‌عنوان یک نیاز پژوهشی و صنعتی مطرح می‌شود.

با گسترش فناوری اطلاعات و افزایش نیاز به انجام محاسبات پیچیده در حوزه‌های مختلف علمی، مهندسی، صنعتی و حتی تجاری، سیستم‌های پردازشی سنتی با محدودیت‌های جدی مواجه شده‌اند. پردازنده‌های مرکزی (CPU) با وجود عملکرد بسیار بالا در پردازش‌های ترتیبی، در مواجهه با بارهای محاسباتی سنگین و قابل موازی‌سازی، بازدهی مطلوبی ندارند. در چنین شرایطی، پردازنده‌های گرافیکی (GPU) به‌عنوان راه‌حلی نوین و کارآمد برای انجام محاسبات هم‌زمان و تسریع فرآیندهای پردازشی مطرح شده‌اند. اهمیت این موضوع از آن جهت برجسته است که بسیاری از الگوریتم‌ها و برنامه‌های کاربردی نوین، نیازمند توان پردازشی بالا در زمان کم هستند.

GPU ها در ابتدا تنها برای پردازش گرافیکی توسعه یافته بودند، اما به‌دلیل ساختار معماری خاص خود، امکان بهره‌برداری در زمینه‌های محاسباتی فراتر از رندرینگ تصویری را نیز فراهم کردند. با ظهور مدل‌های برنامه‌نویسی مانند CUDA و OpenCL، دسترسی توسعه‌دهندگان به قدرت پردازشی GPU آسان‌تر شد و زمینه برای استفاده از آن‌ها در حوزه‌هایی نظیر هوش مصنوعی، یادگیری ماشین، تحلیل داده‌های حجیم،

پردازش تصویر و شبیه‌سازی‌های علمی مهیا گردید. در بسیاری از این حوزه‌ها، استفاده از GPU منجر به کاهش چشمگیر زمان اجرا و افزایش کارایی سیستم شده است، که این مسئله نشان‌دهنده اهمیت راهبردی این فناوری در دنیای امروز است.

اهمیت تحقیق در مورد پردازنده‌های گرافیکی همچنین به دلیل نقش کلیدی آن‌ها در شکل‌دهی به آینده رایانش ابری، اینترنت اشیا، تحلیل بلادرنگ داده‌ها و توسعه زیرساخت‌های هوش مصنوعی است. شرکت‌های بزرگی مانند Google، Amazon، Meta و NVIDIA سرمایه‌گذاری‌های گسترده‌ای روی فناوری GPU انجام داده‌اند که گواهی بر جایگاه حیاتی آن در صنایع نوظهور است. شناخت عمیق‌تری از ساختار، مزایا، چالش‌ها و روش‌های استفاده از GPU، می‌تواند بستری مناسب برای توسعه راهکارهای بومی و ارتقای توانمندی‌های فناورانه در کشور فراهم آورد.

در دنیای امروز، استفاده از توان پردازشی بالا، به‌ویژه در مسائل موازی‌پذیر، یک ضرورت اجتناب‌ناپذیر است این تحقیق تلاش می‌کند با مروری بر مفاهیم پایه، معماری سخت‌افزاری GPU، و مدل‌های برنامه‌نویسی آن، درکی قابل قبول از قابلیت‌ها و ظرفیت‌های این پردازنده‌ها ارائه دهد.

فهرست مطالب:

فصل اول : مبانی اولیه

از ترانزیستور به سوی CPU

ترانزیستورها بلوک‌های سازنده اصلی در الکترونیک دیجیتال مدرن هستند. این دستگاه‌های نیمه‌هادی به‌عنوان سوئیچ عمل می‌کنند و جریان الکتریکی را بر اساس سیگنال‌های ورودی کنترل می‌کنند. در مدارهای دیجیتال، ما عمدتاً از ترانزیستورهای اثر میدان نیمه‌هادی اکسید فلزی (MOSFET) استفاده می‌کنیم که می‌توانند از نوع NMOS یا PMOS باشند. با ترکیب چندین ترانزیستور، می‌توان گیت‌های منطقی را ساخت که واحدهای پایه برای انجام عملیات منطقی در سیستم‌های دیجیتال هستند.

گیت‌های منطقی مانند AND، OR، NOT، NAND، NOR، XOR و XNOR از ترانزیستورها ساخته می‌شوند و عملیات باینری خاصی را انجام می‌دهند. به‌عنوان مثال، گیت AND تنها زمانی خروجی بالا (۱) تولید می‌کند که همه ورودی‌هایش بالا باشند، در حالی که گیت OR اگر حداقل یکی از ورودی‌ها بالا باشد، خروجی بالا می‌دهد. گیت NOT سیگنال ورودی را معکوس می‌کند. این گیت‌ها اساس ساخت مدارهای پیچیده‌تر را تشکیل می‌دهند و امکان اجرای توابع منطقی متنوع را فراهم می‌کنند.

مدارهای ترکیبی مانند دیکودرها و مالتی‌پلکسرها با استفاده از گیت‌های منطقی ساخته می‌شوند و عملکردشان تنها به ورودی‌های فعلی بستگی دارد. دیکودر اطلاعات باینری را از n خط ورودی به حداکثر n^۲ خط خروجی منحصربه‌فرد تبدیل می‌کند؛ مثلاً یک دیکودر ۲ به ۴ با دو ورودی، یکی از چهار خروجی را فعال می‌کند و در آدرس‌دهی حافظه کاربرد دارد. مالتی‌پلکسر یکی از چندین ورودی را انتخاب کرده و به یک خروجی واحد هدایت می‌کند؛ مثلاً یک مالتی‌پلکسر ۴ به ۱ با استفاده از دو خط انتخاب، ورودی موردنظر را مشخص می‌کند. این مدارها با ترکیب گیت‌هایی مثل AND، OR و NOT ساخته می‌شوند.

فلیپ‌فلاپ‌ها به‌عنوان مدارهای ترتیبی وارد بحث می‌شوند که خروجی آن‌ها نه‌تنها به ورودی‌های فعلی، بلکه به حالت‌های قبلی نیز بستگی دارد. فلیپ‌فلاپ‌ها عناصر ذخیره‌سازی پایه هستند و انواع مختلفی مانند SR، D، JK و T دارند. برای مثال، فلیپ‌فلاپ D داده ورودی را در لبه خاصی از سیگنال ساعت ثبت کرده و تا لبه بعدی حفظ می‌کند. این اجزا برای ذخیره یک بیت اطلاعات و همگام‌سازی عملیات در سیستم‌های دیجیتال حیاتی هستند.

رجیسترها با کنار هم قرار دادن چندین فلیپ‌فلاپ ساخته می‌شوند و امکان ذخیره داده‌های چندبیتی را فراهم می‌کنند. به‌عنوان مثال، یک رجیستر ۸ بیتی از ۸ فلیپ‌فلاپ تشکیل شده که هر کدام یک بیت را ذخیره می‌کنند. رجیسترها در پردازنده‌ها برای نگهداری موقت دستورات، آدرس‌ها و داده‌ها در حین پردازش استفاده می‌شوند و نقش مهمی در انتقال و دستکاری داده‌ها دارند.

واحد محاسبه و منطق (ALU) یکی از اجزای اصلی CPU است که عملیات حسابی و منطقی را انجام می‌دهد. این واحد می‌تواند اعمالی مانند جمع، تفریق، ضرب، تقسیم و عملیات بیتی مثل AND، OR و XOR را اجرا کند. ALU از مدارهای ترکیبی مانند جمع‌کننده‌ها، تفریق‌کننده‌ها و گیت‌های منطقی، به همراه مالتی‌پلکسرها برای انتخاب عملیات موردنظر بر اساس سیگنال‌های کنترلی تشکیل شده است.

واحد کنترل (CU) عملیات CPU را هدایت می‌کند و با تولید سیگنال‌های کنترلی، جریان داده بین ALU، رجیسترها و حافظه را مدیریت می‌کند. این واحد دستورات را از حافظه واکشی می‌کند، آن‌ها را رمزگشایی کرده و با هماهنگی اجزای مختلف اجرا می‌کند. CU از ترکیب گیت‌های منطقی، فلیپ‌فلاپ‌ها و سایر مدارها استفاده می‌کند تا توالی عملیات را کنترل کرده و اجرای صحیح دستورات را تضمین کند.

در نهایت، معماری CPU همه این اجزا را در یک واحد پردازشی کامل ادغام می‌کند. این معماری شامل ALU برای محاسبات، CU برای کنترل، رجیسترهای مختلف مانند شمارنده برنامه (PC)، رجیستر دستور (IR) و رجیسترهای عمومی برای ذخیره داده‌ها، و باس‌ها برای انتقال داده است. CPU در سیکلی از واکشی، رمزگشایی و اجرای دستورات عمل می‌کند و با تجزیه وظایف پیچیده به مراحل ساده، عملکرد سیستم را ممکن می‌سازد.

تعاریف مورد نیاز

دنیای سخت افزار با شتابی بی‌سابقه ای در حال پیشرفت است، هر قطعه که تولید می‌شود چندین مفهوم و عملکرد را وارد این وادی میکند و حتی ممکن است این ورود باعث شود مفاهیم و عملکرد های قدیمی تر منسوخ شده و به صفحات تاریخ علم بپیوندند در این راستا ما نیاز داریم تا مفاهیم اولیه و حیاتی در این زمینه را بشناسیم، اصطلاحاتی همچون Thread، CUDA، OpenCL، Render و ... واژگانی هستند که برای درک برای توضیحات آینده ضروری است، از این رو توضیح مختصری از هر یک خواهیم داشت

Thread (رشته یا نخ پردازشی)

Thread یا «نخ» یک واحد مستقل اجرایی درون یک برنامه یا فرآیند (Process) است که می‌تواند به‌طور موازی با سایر نخ‌ها اجرا شود. در یک برنامه چندرشته‌ای (Multithreaded)، چند Thread می‌توانند به‌طور هم‌زمان بر روی یک یا چند هسته‌ی پردازنده اجرا شوند. این روش باعث افزایش کارایی و کاهش زمان اجرای برنامه‌های پیچیده می‌شود. امروزه اکثر CPUها و GPUها برای پشتیبانی از چندنخی (Multithreading) طراحی شده‌اند.

در پردازنده‌های گرافیکی، مفاهیم نخی حتی از اهمیت بیشتری برخوردار است؛ زیرا یک GPU می‌تواند هم‌زمان ده‌ها هزار نخ را اجرا کند. به همین دلیل معماری GPU برای اجرای موازی بسیار گسترده طراحی

شده است. در زبان‌هایی مثل CUDA (برای انویدیا) یا OpenCL (برای AMD)، هر Thread یک وظیفه‌ی کوچک و مستقل را برعهده دارد؛ مثل محاسبه‌ی رنگ یک پیکسل یا شبیه‌سازی حرکت یک ذره در گرافیک.

سیستم‌عامل‌ها و کتابخانه‌های نرم‌افزاری نیز امکاناتی برای مدیریت Threadها فراهم می‌کنند. برای مثال در ویندوز از WinThreads، در لینوکس از Pthreads و در زبان‌های برنامه‌نویسی از Thread Pool، Mutex و Semaphore استفاده می‌شود. بهره‌گیری صحیح از چندرشته‌ای‌سازی می‌تواند عملکرد برنامه را چندین برابر افزایش دهد، اما هم‌زمان چالش‌هایی مثل هم‌زمانی (Race Condition) و انسداد (Deadlock) را به‌همراه دارد که باید مدیریت شوند.

فلاپس (FLOPS)

واژه‌ی FLOPS مخفف Floating Point Operations Per Second است و به تعداد عملیات ممیز شناور که یک پردازنده (CPU یا GPU) در هر ثانیه می‌تواند انجام دهد اشاره دارد. این معیار یکی از مهم‌ترین شاخص‌ها برای سنجش قدرت محاسباتی یک پردازنده است، به‌ویژه در کاربردهای علمی، گرافیکی، و یادگیری ماشین. برخلاف عملیات صحیح (integer)، عملیات ممیز شناور پیچیده‌تر و محاسباتی‌تر هستند و دقت بالاتری دارند.

برای اندازه‌گیری FLOPS، معمولاً از پیشوندهایی مانند GFLOPS (میلیارد عملیات)، TFLOPS (تریلیون عملیات) و PFLOPS (کادریلیون عملیات) استفاده می‌شود. به عنوان مثال، یک کارت گرافیک مدرن مانند AMD Radeon RX 7900 XTX ممکن است تا ۶۰+ TFLOPS قدرت داشته باشد. این به این معناست که در هر ثانیه می‌تواند بیش از ۶۰ تریلیون عملیات ممیز شناور انجام دهد؛ که برای پردازش گرافیک‌های سنگین یا اجرای مدل‌های یادگیری عمیق بسیار حیاتی است.

در مقایسه میان CPU و GPU، FLOPS نقش کلیدی دارد، چرا که GPUها به دلیل داشتن هزاران هسته‌ی پردازشی، در پردازش‌های موازی با حجم زیاد (مانند یادگیری ماشین، رمزنگاری، یا پردازش تصاویر) به مراتب قدرت بیشتری از CPU دارند. بنابراین FLOPS به عنوان واحد مقیاس جهانی در تحلیل عملکرد ابررایانه‌ها، شتاب‌دهنده‌ها، و کارت‌های گرافیکی به‌کار می‌رود.

Coprocessor (کمک‌پردازنده)

کمک‌پردازنده یا Coprocessor یک واحد سخت‌افزاری جانبی است که به پردازنده‌ی اصلی (CPU) کمک می‌کند تا عملیات خاصی را سریع‌تر و کارآمدتر انجام دهد. این مفهوم در ابتدا در دهه ۱۹۸۰ برای واحدهای محاسبات ممیز شناور (FPU) به‌کار رفت، زمانی که CPUها توان پردازش عددی بالایی نداشتند. کمک‌پردازنده‌ها می‌توانند شامل FPU، GPU، TPU، DSP یا هر نوع پردازنده تخصصی دیگری باشند.

یکی از مهم‌ترین و امروزی‌ترین مصداق‌های کمک‌پردازنده، GPU است. GPU در ابتدا فقط برای گرافیک طراحی شده بود، اما اکنون با قابلیت‌های GPGPU به عنوان یک کمک‌پردازنده قدرتمند برای

کارهای محاسباتی، یادگیری عمیق، تحلیل داده و رمزنگاری به‌کار می‌رود. همچنین شرکت‌هایی مانند Google با طراحی TPU (Tensor Processing Unit) نیز کمک‌پردازنده‌هایی مختص یادگیری ماشین ساخته‌اند که نقش بسیار مهمی در آموزش مدل‌های بزرگ دارند.

در بسیاری از سیستم‌های مدرن، به‌ویژه ابررایانه‌ها و مراکز داده، ترکیب CPU و کمک‌پردازنده‌ها به‌صورت هماهنگ باعث افزایش چشمگیر کارایی می‌شود. برای مثال در معماری‌های جدید، داده‌های سنگین ابتدا توسط GPU یا TPU پردازش شده و سپس نتایج به CPU منتقل می‌شود تا تصمیم‌گیری یا تحلیل انجام شود. استفاده بهینه از کمک‌پردازنده‌ها مستلزم طراحی نرم‌افزارهایی با معماری موازی و آگاهی از ظرفیت‌های هر واحد است.

(Compute Unified Device Architecture) CUDA

CUDA یک پلتفرم محاسباتی و مدل برنامه‌نویسی است که توسط شرکت NVIDIA توسعه یافته و امکان استفاده از GPU برای انجام پردازش‌های عمومی (نه فقط گرافیکی) را فراهم می‌کند. با استفاده از CUDA، توسعه‌دهندگان می‌توانند برنامه‌هایی بنویسند که از توان پردازشی هزاران هسته موازی در کارت گرافیک بهره ببرند. این فناوری در زمینه‌هایی چون یادگیری ماشین، شبیه‌سازی‌های علمی، پردازش تصویر و تحلیل داده‌های بزرگ کاربرد گسترده‌ای دارد. مزیت اصلی CUDA، بهره‌گیری مستقیم و بهینه از منابع سخت‌افزاری اختصاصی کارت‌های گرافیک NVIDIA است.

Render (رندر یا پردازش نهایی تصویر)

رندرینگ به فرآیند نهایی‌سازی تصویر یا صحنه‌ی سه‌بعدی و تبدیل آن به یک تصویر دوبعدی قابل‌نمایش گفته می‌شود. این مرحله یکی از حیاتی‌ترین بخش‌های گرافیک رایانه‌ای، بازی‌سازی، فیلم‌سازی و طراحی مهندسی است. رندر می‌تواند به‌صورت بلادرنگ (Real-Time) در بازی‌ها یا به‌صورت آفلاین در تولیدات سینمایی انجام شود. هر دو نوع رندر شامل محاسبات پیچیده‌ی نور، سایه، بافت، بازتاب، شکست نور و... هستند.

در بازی‌ها، رندر توسط پردازنده گرافیکی (GPU) و با استفاده از شیدرها، تکنیک‌های لوله گرافیکی (Graphics Pipeline) و APIهایی مانند DirectX، Vulkan و OpenGL انجام می‌شود. مرحله‌ی رندر معمولاً شامل چند مرحله مهم مانند Geometry Processing، Rasterization، Shading و Compositing است. برای مثال در یک بازی، ابتدا هندسه‌ی صحنه پردازش می‌شود، سپس آن به پیکسل‌های روی صفحه تقسیم می‌گردد، و در نهایت نور، سایه، افکت‌های محیطی و Post Processing اعمال می‌شوند.

در رندر آفلاین (مثل رندر انیمیشن یا معماری)، از تکنیک‌هایی مانند Ray Tracing یا Path Tracing استفاده می‌شود که دقت بالایی دارند اما زمان‌بر هستند. موتورهای رندر مانند V-Ray، Arnold و Blender Cycles از این روش‌ها بهره می‌برند. در سوی مقابل، تکنولوژی‌های جدیدی مانند DLSS (در

انویدیا) و FSR (در AMD) تلاش می‌کنند با استفاده از هوش مصنوعی، رندر را سریع‌تر و کارآمدتر کنند. رندرینگ قلب تولیدات گرافیکی است و کیفیت خروجی بصری مستقیماً به کیفیت و تکنیک‌های رندر بستگی دارد.

شیدر (Shader)

شیدرها در واقع برنامه‌های کوچکی هستند که برای اجرا بر روی پردازنده‌ی گرافیکی (GPU) طراحی شده‌اند و وظیفه‌ی آن‌ها تولید و پردازش جلوه‌های بصری در گرافیک رایانه‌ای است. آن‌ها بخش جدایی‌ناپذیر موتورهای گرافیکی بازی‌ها، نرم‌افزارهای طراحی سه‌بعدی، و حتی رابط‌های گرافیکی سیستم عامل هستند. شیدرها از دهه ۲۰۰۰ به شکل گسترده در کارت‌های گرافیک استفاده شدند و باعث تحول در کیفیت تصویری بازی‌ها و شبیه‌سازی‌های بصری شدند.

شیدرها انواع مختلفی دارند که هرکدام برای وظیفه خاصی طراحی شده‌اند: Vertex Shader (پردازش رأس‌ها)، Pixel/Fragment Shader (پردازش پیکسل‌ها)، Geometry Shader (پردازش اجسام هندسی)، و Compute Shader (پردازش عمومی). شیدرها معمولاً با زبان‌هایی مانند HLSL (در DirectX)، GLSL (در OpenGL)، یا SPIR-V (در Vulkan) نوشته می‌شوند. این برنامه‌ها بر روی هسته‌های شیدر GPU اجرا می‌شوند که معمولاً بسیار زیاد و موازی هستند و به پردازش سریع حجم انبوهی از داده‌های گرافیکی کمک می‌کنند.

با پیشرفت فناوری گرافیک، شیدرها فقط به جلوه‌های بصری محدود نشدند، بلکه به ابزارهایی برای محاسبات عمومی روی GPU نیز تبدیل شدند (GPGPU). برای مثال در یادگیری ماشین، شبیه‌سازی فیزیکی، و رمزنگاری می‌توان از قابلیت Compute Shader برای اجرای کدهای موازی استفاده کرد. همین انعطاف‌پذیری، GPU را از یک ابزار صرفاً گرافیکی به یک ماشین محاسباتی چندمنظوره تبدیل کرده است.

HPC (High Performance Computing)

عبارت HPC (پردازش با کارایی بالا) به مجموعه‌ای از فناوری‌ها و زیرساخت‌ها اطلاق می‌شود که برای اجرای برنامه‌های محاسباتی بسیار سنگین طراحی شده‌اند؛ مانند شبیه‌سازی‌های علمی، مدل‌سازی آب‌وهوا، تحلیل داده‌های ژنومی، و آموزش مدل‌های هوش مصنوعی. سیستم‌های HPC معمولاً شامل مجموعه‌ای از پردازنده‌های مرکزی (CPU)، پردازنده‌های گرافیکی (GPU)، حافظه‌های سریع، و شبکه‌های ارتباطی با سرعت بالا هستند.

در دنیای HPC، هزاران نود (node) کامپیوتری به‌صورت موازی فعالیت می‌کنند تا بار محاسباتی عظیم را بین خود تقسیم کرده و در مدت‌زمان کوتاه‌تری به نتیجه برسند. برای مثال، ابررایانه‌هایی مانند Frontier یا El Capitan که با پردازنده‌ها و GPUهای AMD ساخته شده‌اند، به توان پردازشی در حد

ExaFLOPS (۱۸^۱۰ عملیات در ثانیه) دست یافته‌اند. این سیستم‌ها در صنایع فضایی، پزشکی، انرژی و فیزیک کاربرد دارند.

یکی از نکات کلیدی در HPC، کارایی به ازای وات است. به همین دلیل معماری‌هایی مانند CDNA در AMD یا Hopper در NVIDIA با هدف ارائه حداکثر عملکرد با کمترین مصرف انرژی طراحی می‌شوند. همچنین، فناوری‌هایی مانند Infinity Fabric در AMD، یا NVLink در NVIDIA برای اتصال سریع بین چیپ‌ها یا نودها به کار گرفته می‌شوند تا پهنای باند و کارایی سیستم افزایش یابد. HPC آینده‌ی تحلیل داده‌های بزرگ و هوش مصنوعی است و یکی از زمینه‌های رقابت اصلی شرکت‌های بزرگ نیمه‌رسانا به شمار می‌آید.

Real-time (بلا درنگ)

مفهوم Real-time یا «بلادرنگ» در علوم رایانه‌ای و به‌ویژه در حوزه‌ی گرافیک و پردازش بازی‌ها، به سیستمی اطلاق می‌شود که باید وظایف خود را در محدوده‌ی زمانی مشخص و بسیار کوتاهی انجام دهد، به‌گونه‌ای که پاسخ آن بلافاصله برای کاربر قابل مشاهده یا حس باشد. برخلاف پردازش‌های سنتی که می‌توانند زمان‌بر باشند و اولویت آن‌ها دقت یا کیفیت است، در سیستم‌های بلادرنگ، زمان پاسخ‌دهی مهم‌ترین فاکتور است؛ زیرا حتی در صورت درست بودن نتیجه، اگر با تأخیر ارائه شود، بی‌ارزش خواهد بود.

در گرافیک بازی‌ها، Real-time Rendering به فرآیندی گفته می‌شود که در آن، تمام تصاویر، افکت‌ها، حرکات دوربین، تعاملات کاربر و نورپردازی‌ها در کسری از ثانیه و در همان لحظه‌ای که کاربر نیاز دارد، تولید و نمایش داده می‌شوند. این یعنی یک سیستم گرافیکی باید بتواند حداقل ۳۰ تا ۶۰ فریم در ثانیه (fps) را پردازش و نمایش دهد تا بازی برای کاربر روان، طبیعی و قابل تجربه باشد. هر فریم باید در حدود ۱۶ تا ۳۳ میلی‌ثانیه محاسبه و آماده شود، که این خود نیازمند سخت‌افزار بسیار بهینه، الگوریتم‌های سریع و ساختار داده‌های مناسب است.

کاربرد Real-time محدود به بازی‌ها نیست و در حوزه‌های دیگری مانند سیستم‌های کنترل صنعتی، شبیه‌سازی پرواز، سیستم‌های نظامی، خودروهای خودران و واقعیت مجازی (VR) نیز استفاده می‌شود. در این موارد، تصمیمات و واکنش‌های سیستم باید آنی باشند، چرا که هرگونه تأخیر ممکن است باعث ایجاد خطر یا اختلال شود. در بازی‌ها، تأخیر می‌تواند باعث کاهش کیفیت تجربه‌ی کاربری شود، اما در سیستم‌های حیاتی مانند کنترل پرواز یا پزشکی، یک تأخیر کوچک می‌تواند پیامدهای جدی و حتی مرگ‌بار داشته باشد. بنابراین، طراحی سیستم‌های Real-time، نیازمند توازن بسیار دقیقی بین سرعت، دقت، و منابع سخت‌افزاری است.

چیپلت (Chiplet)

چیپلت به روشی نوین در طراحی تراشه‌ها اشاره دارد که به‌جای ساخت یک تراشه‌ی بزرگ و یکپارچه، چندین تراشه‌ی کوچک‌تر (چیپلت) روی یک پکیج قرار می‌گیرند تا یک واحد پردازشی قدرتمند تشکیل

دهند. این روش برای اولین بار توسط AMD در طراحی پردازنده‌های سری Ryzen و سپس در کارت‌های گرافیک سری RDNA 3 به کار گرفته شد. هدف از این رویکرد، افزایش بهره‌وری، کاهش هزینه ساخت، و بهبود عملکرد است.

مزیت اصلی چیپلت در این است که تراشه‌های کوچکتر به‌سادگی روی ویفر سیلیکونی تولید می‌شوند، نرخ خرابی پایین‌تری دارند، و فرآیند تولید آن‌ها می‌تواند از فناوری‌های ساخت متفاوتی بهره‌مند شود. به‌عنوان مثال، ممکن است چیپلت گرافیکی با فناوری ۵ نانومتری ساخته شود در حالی که چیپلت کنترلر یا کش با فناوری ۶ یا ۷ نانومتری تولید گردد. این موضوع باعث انعطاف بیشتر در طراحی و صرفه‌جویی اقتصادی می‌شود.

در زمینه‌ی GPU، AMD با استفاده از چیپلت در معماری RDNA 3 توانست کارت‌هایی مانند RX 7900 XTX را با ترکیب چیپلت اصلی (Graphics Compute Die) و چیپلت حافظه (Memory Cache Die) ارائه کند. این کار، ضمن افزایش کش و پهنای باند داخلی، موجب شد که AMD به‌جای طراحی یک GPU عظیم، از چند قطعه کوچکتر استفاده کند و همچنان کارایی بالایی به‌دست آورد. چیپلت‌ها آینده‌ی طراحی تراشه‌ها محسوب می‌شوند و نقش مهمی در کاهش هزینه، افزایش مقیاس‌پذیری، و کاهش گرما خواهند داشت

تاریخچه ای از ظهور کارت گرافیک

در نخستین سال‌های ظهور رایانه‌های دیجیتال، پردازنده‌های مرکزی (Central Processing Units یا CPUها) به‌عنوان تنها واحد پردازشی سیستم شناخته می‌شدند. این پردازنده‌ها طراحی شده بودند تا وظایف عمومی محاسباتی را به‌صورت ترتیبی و مرحله‌به‌مرحله اجرا کنند. معماری آن‌ها معمولاً شامل تعداد محدودی هسته، حافظه کش، و واحدهای منطقی حسابی (ALU) بود و برای اجرای برنامه‌های معمولی، نظیر سیستم‌های عامل، پردازش متن و برنامه‌های اداری، کفایت می‌کرد. افزایش توان پردازشی در CPUها ابتدا از طریق افزایش پالس ساعت صورت گرفت، اما محدودیت‌های فیزیکی مانند تولید گرما، مصرف انرژی و اشباع قانون مور باعث شد که توسعه به سمت استفاده از چند هسته پردازشی (Multi-Core CPUs) حرکت کند.

در دهه ۱۹۹۰ میلادی، با گسترش نیاز به گرافیک‌های پیشرفته‌تر، طراحی کارت‌های گرافیک تخصصی مورد توجه قرار گرفت. پردازنده‌های گرافیکی (GPU) در ابتدا به عنوان پردازنده‌هایی تخصصی طراحی شدند که وظیفه اصلی آن‌ها دستکاری و تغییر سریع حافظه برای افزایش نرخ ساخت تصاویر در یک فریم بود. هدف اولیه این واحدها تسریع فرآیند ساخت تصویر برای خروجی نمایشگر بود و به طور گسترده به عنوان معماری‌های چندرشته‌ای (multi-threaded) و موازی انبوه (massively multi-threaded) برای محاسبات گرافیکی و بصری (2D/3D) مورد استفاده قرار می‌گرفتند. شرکت NVIDIA در سال ۱۹۹۹ با

معرفی GeForce 256، اصطلاح "GPU" را رواج داد و آن را به عنوان "اولین GPU جهان" معرفی کرد که موتورهای تبدیل، تنظیم/برش مثلث، نورپردازی و رندرینگ را یکپارچه کرده بود. در اوایل دهه ۱۹۹۰، با ظهور بازی‌های سه‌بعدی، نیاز به سخت‌افزارهای شتاب‌دهنده سه‌بعدی پدیدار شد و GPUها در ابتدا پردازنده‌هایی گران‌قیمت برای ایستگاه‌های کاری با عملکرد بالا بودند که وظیفه کاهش بار محاسباتی گرافیکی از پردازنده مرکزی (CPU) را بر عهده داشتند. در مراحل اولیه، GPUها پردازنده‌هایی با عملکرد ثابت (fixed-function) بودند که عمدتاً برای رندرینگ گرافیک سه‌بعدی استفاده می‌شدند و خط لوله اجرایی آن‌ها قابل تنظیم بود اما کاملاً برنامه‌پذیر نبود. استفاده از چندین کارت گرافیک یا تعداد زیادی تراشه گرافیکی، ماهیت موازی پردازش گرافیک را بیش از پیش موازی می‌ساخت.

با افزایش قابلیت برنامه‌پذیری، GPUها فراتر از محاسبات گرافیکی، قادر به انجام وظایف عمومی شدند و به دلیل سرعت بالایشان به عنوان کمک‌پردازنده‌هایی (coprocessors) مفید برای کاربردهای متنوعی تبدیل گشتند. مفهوم "محاسبات عمومی بر روی پردازنده‌های گرافیکی" (GPGPU) به دلیل افزایش سریع عملکرد سخت‌افزار گرافیکی و بهبود قابلیت برنامه‌پذیری آن پدیدار شد. در اوایل قرن ۲۱، خطوط لوله GPGPU ابتدا برای بهبود شیدرهای گرافیکی توسعه یافتند، اما به زودی مشخص شد که برای نیازهای محاسبات علمی نیز مناسب هستند. یک نقطه عطف مهم در سال ۲۰۰۳ بود که دو گروه تحقیقاتی به طور مستقل رویکردهای مبتنی بر GPU را برای حل مسائل جبر خطی عمومی کشف کردند که سریع‌تر از CPUها عمل می‌کرد. معرفی چارچوب معماری یکپارچه دستگاه محاسباتی (CUDA) توسط NVIDIA در سال ۲۰۰۶ یک تغییردهنده بازی بود. CUDA یک مدل برنامه‌نویسی و ابزارهایی را فراهم کرد که به توسعه‌دهندگان اجازه می‌داد از قابلیت‌های پردازش موازی GPUها برای محاسبات عمومی، از جمله محاسبات علمی، شبیه‌سازی‌ها و در نهایت کاربردهای هوش مصنوعی استفاده کنند. GPUها تا جایی تکامل یافته‌اند که بسیاری از کاربردهای دنیای واقعی به طور قابل توجهی سریع‌تر از سیستم‌های چند هسته‌ای بر روی آن‌ها اجرا می‌شوند و مزیت قیمت به عملکرد خوبی را ارائه می‌دهند. آن‌ها می‌توانند به عنوان یک راه‌حل جایگزین یا مکمل برای سرورهای چند هسته‌ای استفاده شوند.

تکامل GPUها از پردازنده‌های صرفاً گرافیکی به واحدهای محاسباتی عمومی، به طور قابل توجهی تحت تأثیر تقاضاهای رو به رشد برای گرافیک‌های پیچیده‌تر و واقع‌گرایانه‌تر قرار گرفت. نیاز به رندرینگ سریع و همزمان میلیون‌ها پیکسل و هندسه پیچیده در بازی‌های سه‌بعدی و کاربردهای بصری، منجر به توسعه معماری‌های موازی انبوه با هزاران هسته ساده و تخصصی شد که برای عملیات ممیز شناور موازی بهینه شده بودند. این معماری، که در ابتدا پشت خطوط لوله ثابت پنهان بود، با گذشت زمان به طور فزاینده‌ای قابل برنامه‌ریزی شد. این قابلیت برنامه‌ریزی، همراه با شدت محاسباتی بالای وظایف گرافیکی، به طور ناخواسته یک موتور قدرتمند برای هر نوع محاسبه موازی و داده‌محور، فراتر از گرافیک، ایجاد کرد. کشف اینکه این خطوط لوله گرافیکی می‌توانند محاسبات علمی را شتاب دهند ، یک مسیر تکاملی حیاتی را آشکار می‌سازد: تقاضاهای شدید گرافیک، نوآوری سخت‌افزاری را در مسیری سوق داد که به طور اتفاقی پتانسیل‌های

جدیدی را برای محاسبات موازی عمومی باز کرد. این نشان می‌دهد که پیشرفت‌های تکنولوژیکی اغلب کاربردهای پیش‌بینی نشده‌ای فراتر از هدف اصلی خود دارند که توسط کارایی‌های معماری زیربنایی هدایت می‌شوند.

پذیرش گسترده GPGPU نه تنها یک دستاورد فنی، بلکه یک پدیده بازارمحور نیز بوده است. منابع متعدد به "مزیت قیمت به عملکرد خوب" ، "هزینه‌های پایین" ، "نسبت قیمت/عملکرد جذاب" و "قدرتمندترین سخت‌افزار محاسباتی به ازای هر دلار" اشاره می‌کنند. این مسئله صرفاً در مورد سرعت خام نیست، بلکه در مورد کارایی آن سرعت نسبت به هزینه است. پردازنده‌های مرکزی (CPU) با وجود تطبیق‌پذیری، برای وظایف بسیار موازی و از نظر ریاضی فشرده که GPUها در آن‌ها برتری دارند، طراحی نشده بودند. شتاب سریع عملکرد GPU، که به طور متوسط هر شش ماه دو برابر می‌شد و از قانون مور برای CPUها پیشی می‌گرفت ، آن‌ها را به یک جایگزین اقتصادی جذاب تبدیل کرد. این نشان می‌دهد که مزیت اقتصادی سخت‌افزار موازی تخصصی برای بارهای کاری خاص، عامل مهمی در پذیرش گسترده GPGPU بوده است. این روند با افزایش کارایی انرژی GPUها برای دستیابی به عملکرد بالا ادامه یافته است.

فصل دوم: پردازش موازی

پردازش موازی

در روش‌های سنتی پردازش، اجرای دستورات به‌صورت ترتیبی و پشت‌سرهم توسط یک پردازنده مرکزی (CPU) انجام می‌شود اما این رویکرد در مواجهه با مسائل بزرگ و زمان‌بر، محدودیت‌هایی از نظر توان محاسباتی، مصرف انرژی و بازدهی دارد. پردازش موازی (Parallel Processing) رویکردی در علوم کامپیوتر است که در آن چندین عملیات یا محاسبات به طور همزمان انجام می‌شوند. هدف اصلی این روش، افزایش سرعت و کارایی سیستم‌های کامپیوتری با تقسیم یک وظیفه بزرگ به چندین وظیفه کوچکتر و اجرای همزمان آنها است. به جای اینکه یک پردازنده واحد تمام کار را مرحله به مرحله انجام دهد، چندین واحد پردازشی (که می‌توانند هسته‌های یک پردازنده، پردازنده‌های جداگانه، یا حتی کامپیوترهای مختلف در یک شبکه باشند) به صورت هماهنگ بر روی بخش‌های مختلفی از مسئله کار می‌کنند. پردازش موازی می‌تواند در سطوح مختلفی پیاده‌سازی شود

- پردازش موازی در سطح بیت (Bit-level Parallelism):

پردازش موازی در سطح بیت ساده‌ترین و ابتدایی‌ترین نوع موازی‌سازی است که مستقیماً به ساختار داخلی واحدهای پردازنده مرتبط می‌شود. در این مدل، با افزایش اندازه ثبات‌ها (Registers)، واحدهای محاسبه و مسیرهای داده، پردازنده قادر می‌شود چند بیت از یک داده را به‌صورت هم‌زمان پردازش کند. به عنوان مثال، زمانی که یک پردازنده ۳۲ بیتی به یک پردازنده ۶۴ بیتی ارتقاء می‌یابد، قادر است دو عدد ۳۲ بیتی را در یک چرخه زمانی واحد پردازش کند که در پردازنده قبلی به دو چرخه نیاز داشت. این افزایش پهنای باند داده در واحد زمان باعث بهبود کارایی در سطح سخت‌افزار می‌شود، بدون آن‌که نیاز به تغییر در الگوریتم‌های سطح بالا وجود داشته باشد.

- پردازش موازی در سطح دستورالعمل (Instruction-level Parallelism - ILP):

پردازش موازی در سطح دستورالعمل مفهومی در طراحی معماری پردازنده‌هاست که بر اساس آن، چندین دستور مستقل می‌توانند به‌صورت هم‌زمان (یا با هم‌پوشانی زمانی) توسط واحدهای مختلف اجرا شوند. پردازنده‌های مدرن از تکنیک‌هایی مانند پایپ‌لاینینگ (Pipelining)، اجرای خارج از ترتیب (Out-of-Order Execution) و حدس زدن شاخه‌ها (Branch Prediction) استفاده می‌کنند تا دستورالعمل‌های بیشتری را در هر چرخه ساعت اجرا کنند. به عنوان مثال، در حالی که یک دستور در

مرحله جمع‌آوری داده است، دستور بعدی می‌تواند در حال رمزگشایی (Decode) باشد و دستور دیگر در حال اجرا. این نوع موازی‌سازی توسط سخت‌افزار مدیریت می‌شود و در اغلب موارد، برنامه‌نویس دخالت مستقیم ندارد.

- پردازش موازی در سطح داده (Data-level Parallelism - DLP):

پردازش موازی در سطح داده زمانی به‌کار گرفته می‌شود که عملیات یکسانی باید روی مجموعه‌ای بزرگ از داده‌ها انجام شود. در این مدل، داده‌ها به بخش‌های کوچک‌تر تقسیم شده و هر بخش به‌طور موازی توسط واحدهای پردازشی مشابه پردازش می‌شود. این نوع موازی‌سازی برای عملیات‌های عددی، پردازش تصویر، شبیه‌سازی‌های علمی و یادگیری ماشین بسیار مناسب است. معماری‌های SIMD (Single Instruction, Multiple Data)، مانند آنچه در پردازنده‌های گرافیکی (GPU) یافت می‌شود، نمونه بارز پیاده‌سازی DLP هستند؛ برای مثال، زمانی که هزاران پیکسل تصویر به‌طور هم‌زمان با یک فیلتر خاص پردازش می‌شوند.

- پردازش موازی در سطح وظیفه (Task-level Parallelism - TLP):

در پردازش موازی در سطح وظیفه، تمرکز بر اجرای هم‌زمان چندین وظیفه یا Thread مستقل است که هر یک می‌تواند نوع متفاوتی از عملیات را انجام دهد. برخلاف DLP که بر داده‌های مشابه با عملیات یکسان متمرکز است، TLP به اجرای موازی بخش‌های مختلف یک برنامه که قابلیت اجرا به صورت مستقل دارند می‌پردازد. برای مثال، در یک نرم‌افزار ویرایش ویدیو، ممکن است یک Thread به فشرده‌سازی ویدیو بپردازد، Thread دیگر صدا را پردازش کند و Thread سومی برای ذخیره‌سازی فایل خروجی مسئول باشد. این نوع موازی‌سازی معمولاً توسط برنامه‌نویس و با استفاده از کتابخانه‌هایی مانند OpenMP، POSIX Threads یا قابلیت‌های Threading در زبان‌های برنامه‌نویسی پیاده‌سازی می‌شود.

مقایسه پردازش ترتیبی و موازی

در قلب هر سیستم کامپیوتری، روشی برای اجرای دستورالعمل‌ها و انجام محاسبات وجود دارد. دو رویکرد اصلی که دنیای محاسبات را شکل داده‌اند، پردازش ترتیبی (Sequential Processing) و پردازش موازی (Parallel Processing) هستند. هر یک از این روش‌ها فلسفه، مزایا و محدودیت‌های خاص خود را دارند و درک تفاوت‌های کلیدی آن‌ها برای هر علاقه‌مند به علوم کامپیوتر ضروری است. در حالی که پردازش ترتیبی رویکردی کلاسیک و سرراست است، پردازش موازی پاسخی به نیاز روزافزون به سرعت و کارایی در عصر داده‌های عظیم و الگوریتم‌های پیچیده محسوب می‌شود.

پردازش ترتیبی که به آن پردازش سری نیز گفته می‌شود، ابتدایی‌ترین و سنتی‌ترین مدل اجرای دستورالعمل‌ها است. در این مدل، تمام وظایف و عملیات یکی پس از دیگری و به ترتیبی دقیق اجرا

می‌شوند. به زبان ساده، تا زمانی که یک وظیفه به طور کامل به اتمام نرسد، وظیفه بعدی آغاز نخواهد شد. این فرآیند شبیه به یک خط تولید تک‌نفره است که در آن یک کارگر تمام مراحل ساخت یک محصول را به تنهایی و پشت سر هم انجام می‌دهد.

پیچیدگی زمانی اجرای یک برنامه در این نوع پردازش برابر با مجموع زمان‌های لازم برای اجرای تمام دستورالعمل هاست زیرا هر دستورالعمل به صورت گام به گام و در یک توالی از پیش تعیین‌شده اجرا می‌شود اما طراحی، برنامه‌نویسی و خطایابی برنامه‌های ترتیبی معمولاً ساده‌تر است، زیرا جریان کنترل واضح و قابل پیش‌بینی است. به لحاظ کاربردی پردازش ترتیبی برای اکثر وظایف روزمره و عمومی کامپیوتر که وابستگی شدید به مراحل قبلی دارند، کافی و کارآمد است.

پردازش موازی رویکردی است که در آن چندین عملیات یا محاسبات به طور همزمان انجام می‌شوند. هدف اصلی این روش، شکستن یک وظیفه بزرگ به چندین وظیفه کوچکتر و مستقل و سپس اجرای همزمان این وظایف بر روی چندین واحد پردازشی است. این مدل را می‌توان به یک تیم بزرگ از کارگران تشبیه کرد که هر یک بخشی از یک پروژه بزرگ را به طور همزمان پیش می‌برند.

در این نوع پردازش چندین دستورالعمل یا وظیفه در یک لحظه یا در بازه‌های زمانی همپوشان اجرا می‌شوند. با تقسیم کار، پیچیدگی زمانی اجرای یک برنامه به طور چشمگیری کاهش می‌یابد، به خصوص برای مسائل بزرگ و قابل تقسیم. اما بر خلاف روش قبل طراحی، برنامه‌نویسی، هماهنگی و خطایابی برنامه‌های موازی به دلیل نیاز به مدیریت همزمانی، تبادل داده و جلوگیری از تداخل‌ها پیچیده‌تر است.

پردازش ترتیبی و موازی، دو روی یک سکه در دنیای محاسبات هستند. در حالی که پردازش ترتیبی سادگی و قابلیت اطمینان را برای وظایف روزمره فراهم می‌کند، پردازش موازی نیروی محرکه پشت پیشرفت‌های عظیم در علم، فناوری و صنعت در دهه‌های اخیر بوده است. انتخاب بین این دو رویکرد بستگی به ماهیت مسئله، حجم داده‌ها، و الزامات عملکردی دارد. در دنیای فزاینده پیچیده و داده‌محور امروز، توانایی استفاده از قدرت پردازش موازی نه تنها یک مزیت، بلکه یک ضرورت برای حل چالش‌های آینده و گشودن افق‌های جدید در دنیای دیجیتال است.

انواع معماری‌های پردازش موازی

در تاریخ معماری کامپیوتر، تلاش‌های زیادی برای دسته‌بندی و درک سیستم‌های پردازشی مختلف صورت گرفته است. یکی از تأثیرگذارترین و شناخته‌شده‌ترین این دسته‌بندی‌ها، مدل فلین (Flynn’s Taxonomy) است که توسط مایکل جی. فلین در سال ۱۹۷۲ ارائه شد. این مدل، سیستم‌های کامپیوتری را بر اساس دو معیار کلیدی دسته‌بندی می‌کند: تعداد جریان‌های دستورالعمل (Instruction Stream) و تعداد جریان‌های داده (Data Stream) که یک سیستم قادر به پردازش همزمان آن‌هاست. مدل فلین نه

تنها چارچوبی برای تحلیل معماری‌های موجود ارائه می‌دهد، بلکه راهنمایی برای طراحی سیستم‌های موازی آینده نیز محسوب می‌شود.

فلین سیستم‌ها را بر اساس وجود یک یا چند جریان دستورالعمل و یک یا چند جریان داده طبقه‌بندی می‌کند.

- جریان دستورالعمل (Instruction Stream - IS): به توالی دستورالعمل‌هایی گفته می‌شود که برای انجام یک کار خاص به پردازنده ارسال می‌شوند.
- جریان داده (Data Stream - DS): به توالی داده‌هایی اشاره دارد که توسط دستورالعمل‌ها پردازش می‌شوند.

با ترکیب این دو معیار، چهار دسته‌بندی اصلی در مدل فلین شکل می‌گیرد:

۱. SISD (Single Instruction, Single Data) (یک دستورالعمل، یک داده)

این ساده‌ترین و سنتی‌ترین دسته است که شامل سیستم‌هایی با یک واحد کنترل (برای واکشی و رمزگشایی دستورالعمل‌ها) و یک واحد پردازش (برای اجرای دستورالعمل‌ها) می‌شود. در هر لحظه، تنها یک دستورالعمل روی یک قلم داده اجرا می‌شود. این مدل همان پردازش ترتیبی است.

۲. SIMD (Single Instruction, Multiple Data) - (یک دستورالعمل، چندین داده)

در این معماری، یک واحد کنترل واحد، یک دستورالعمل واحد را به چندین واحد پردازش ارسال می‌کند. هر واحد پردازش این دستورالعمل را به طور همزمان بر روی مجموعه داده‌های متفاوت خود اجرا می‌کند. این مدل برای مسائلی که نیاز به اعمال یک عملیات یکسان بر روی حجم زیادی از داده‌ها دارند، بسیار کارآمد است.

۳. MISD (Multiple Instruction, Single Data) (چندین دستورالعمل، یک داده)

این دسته در عمل کمتر رایج است و تعداد کمی از معماری‌های عملی در این گروه جای می‌گیرند. در MISD، چندین واحد پردازشگر به طور همزمان دستورالعمل‌های متفاوتی را بر روی یک جریان داده واحد اجرا می‌کنند.

٤. MIMD (Multiple Instruction, Multiple Data) - (چندین دستورالعمل، چندین داده)

این انعطاف‌پذیرترین و رایج‌ترین دسته از معماری‌های پردازش موازی مدرن است. در MIMD، چندین واحد پردازش مستقل وجود دارند که هر یک می‌توانند دستورالعمل‌های متفاوتی را به طور همزمان بر روی جریان‌های داده متفاوت خود اجرا کنند. این اجازه می‌دهد تا وظایف کاملاً مستقل از یکدیگر به صورت موازی انجام شوند.

علاوه بر دسته‌بندی فلین، معماری‌های پردازش موازی اغلب بر اساس نحوه اشتراک‌گذاری حافظه و ارتباط بین پردازنده‌ها نیز دسته‌بندی می‌شوند

۱. سیستم‌های حافظه مشترک (Shared Memory Systems)

در این معماری، تمام پردازنده‌ها به یک فضای حافظه فیزیکی مشترک دسترسی دارند. این باعث می‌شود تبادل داده بین پردازنده‌ها بسیار سریع و کارآمد باشد، زیرا نیازی به ارسال پیام‌های صریح نیست. این معماری خود دو نوع دارد نوع اول آن SMP (Symmetric Multiprocessing) است که در آن تمام پردازنده‌ها به طور یکسان به حافظه مشترک دسترسی دارند و زمان دسترسی به هر قسمت از حافظه برای همه پردازنده‌ها تقریباً یکسان است و نوع دوم آن NUMA (Non-Uniform Memory Access) است که هر پردازنده دارای بخش اختصاصی حافظه خود است که دسترسی به آن سریع‌تر از دسترسی به حافظه متعلق به پردازنده‌های دیگر است. این مدل مقیاس‌پذیری بیشتری نسبت به SMP دارد.

۲. سیستم‌های حافظه توزیع‌شده (Distributed Memory Systems)

در این معماری، هر واحد پردازشی (که می‌تواند یک کامپیوتر کامل باشد) دارای حافظه محلی و اختصاصی خود است. برای تبادل داده بین پردازنده‌ها، پیام‌ها باید به صراحت از طریق یک شبکه ارتباطی (مانند اترنت یا اینفینی‌بند) ارسال و دریافت شوند.

۳. سیستم‌های هیبرید (Hybrid Systems)

اکثر سیستم‌های موازی مدرن از ترکیبی از معماری‌های حافظه مشترک و توزیع‌شده استفاده می‌کنند. به عنوان مثال، یک ابرکامپیوتر ممکن است از چندین گره (Node) تشکیل شده باشد که هر گره خود یک سیستم حافظه مشترک (چندین پردازنده چند هسته‌ای) است. ارتباط بین گره‌ها از طریق حافظه توزیع‌شده و ارتباط بین هسته‌ها در یک گره از طریق حافظه مشترک صورت می‌گیرد.

مدل فلین و دسته‌بندی معماری‌های پردازش موازی، ابزارهای تحلیلی قدرتمندی هستند که به ما در درک پیچیدگی و تنوع سیستم‌های محاسباتی کمک می‌کنند. از پردازنده‌های تک‌هسته‌ای SISD گرفته تا ابرکامپیوترهای MIMD که از ترکیبی از حافظه مشترک و توزیع‌شده استفاده می‌کنند، هر معماری دارای نقاط قوت و ضعف خاص خود است و برای کلاس خاصی از مسائل بهینه شده است. با پیشرفت فناوری، مرزهای بین این دسته‌ها ممکن است کمرنگ‌تر شوند، اما اصول بنیادین ارائه شده توسط مدل فلین همچنان به عنوان یک راهنمای اساسی در طراحی و ارزیابی سیستم‌های پردازشی موازی مدرن باقی خواهد ماند.

فصل سوم: ساختار داخلی GPU

مقدمه‌ای بر معماری GPU

در دنیای امروز که داده‌ها با حجم و سرعتی بی‌سابقه تولید و تحلیل می‌شوند، نیاز به پردازش‌های سریع، گسترده و هم‌زمان از همیشه پررنگ‌تر شده است. این نیاز، به‌ویژه در حوزه‌هایی مانند بازی‌های رایانه‌ای، یادگیری ماشین، تحلیل داده‌های بزرگ، رمزنگاری، رندرینگ گرافیکی، و شبیه‌سازی‌های علمی، به شکل بسیار جدی مطرح می‌شود. در چنین شرایطی، پردازنده‌های گرافیکی (GPU) به‌عنوان ابزاری توانمند برای پردازش‌های موازی به صحنه آمده‌اند و نقشی حیاتی در توسعه و شتاب‌دهی به فناوری‌های نوین ایفا می‌کنند.

پردازنده‌های گرافیکی در ابتدا صرفاً برای پردازش گرافیکی طراحی شده بودند. هدف اصلی آن‌ها تسریع در تولید و نمایش تصاویر پیچیده سه‌بعدی در بازی‌های ویدیویی و نرم‌افزارهای گرافیکی بود. اما با گذشت زمان و پیشرفت چشمگیر معماری این پردازنده‌ها، از یک ابزار تخصصی گرافیکی به یک بستر عمومی برای انجام محاسبات با کارایی بالا (High Performance Computing - HPC) بدل شدند. این تغییر بنیادین با معرفی زبان‌ها و رابط‌های برنامه‌نویسی مانند CUDA توسط NVIDIA و OpenCL به وقوع پیوست و زمینه‌ای فراهم کرد تا پژوهشگران، مهندسان، تحلیل‌گران داده و توسعه‌دهندگان نرم‌افزار از قدرت بی‌نظیر GPU برای اجرای انواع الگوریتم‌ها بهره بگیرند.

اما برای درک چرایی و چگونگی این تحول، ابتدا باید به بررسی دقیق‌تر معماری داخلی GPU بپردازیم؛ اینکه چگونه این پردازنده‌ها طراحی شده‌اند، چه اجزایی دارند، چگونه پردازش‌ها را مدیریت می‌کنند، و تفاوت آن‌ها با معماری سنتی CPU در چیست. در معماری GPU برخلاف CPU، که برای اجرای دستورالعمل‌های پیچیده به‌صورت ترتیبی طراحی شده، هدف اصلی اجرای تعداد بسیار زیادی از دستورالعمل‌های ساده به‌صورت هم‌زمان (با استفاده از هزاران هسته سبک‌وزن) است.

این پردازنده‌ها دارای واحدهای پردازشی موازی متعدد، زنجیره‌های حافظه با دسترسی‌های گوناگون، و سامانه‌های زمان‌بندی بسیار دقیق هستند. GPUها معمولاً شامل صدها یا هزاران هسته موازی‌اند که به‌صورت گروهی (مانند Thread و Warp) دستوراتی مشابه را روی داده‌های مختلف اجرا می‌کنند. این مدل از اجرا که به نام SIMT (Single Instruction, Multiple Threads) شناخته می‌شود، قدرت بی‌نظیری در پردازش داده‌های عظیم با الگوریتم‌های تکراری ایجاد می‌کند. ساختار سلسله‌مراتبی حافظه در GPU نیز نقش بسیار مهمی در بهینه‌سازی سرعت و کارایی پردازش دارد، به‌گونه‌ای که طراحی دقیق آن می‌تواند مانع از ایجاد گلوگاه‌های داده‌ای شود.

همچنین معماری GPU طی سال‌های اخیر با افزودن واحدهای اختصاصی مانند Tensor Cores (برای یادگیری عمیق)، RT Cores (برای ردیابی پرتو در گرافیک سه‌بعدی) و Video Encode/Decode Engines برای پردازش چندرسانه‌ای، به یکی از پیچیده‌ترین و چندمنظوره‌ترین ساختارهای سخت‌افزاری در دنیای کامپیوتر تبدیل شده است.

در این فصل، با بررسی دقیق اجزای اصلی معماری GPU، مقایسه آن با CPU، نحوه‌ی مدیریت حافظه و threadها، معماری‌های مختلف در برندهای مختلف (NVIDIA، AMD و Intel)، و کاربردهای متنوع آن در علوم مختلف، گام‌به‌گام به درک عمیق‌تری از قدرت و کارایی این پردازنده‌ها خواهیم رسید. این شناخت به ما کمک می‌کند تا درک کنیم چرا GPU در عصر محاسبات ابری، هوش مصنوعی، و داده‌های بزرگ تا این حد مهم شده، و چگونه می‌توان از آن برای حل چالش‌های پیچیده محاسباتی بهره‌برداری کرد.

مقایسه معماری GPU و CPU

معماری داخلی پردازنده مرکزی (CPU) یکی از پیچیده‌ترین و در عین حال حیاتی‌ترین موضوعات در علوم رایانه و مهندسی سخت‌افزار است. درک دقیق این معماری به ما کمک می‌کند تا بفهمیم چگونه رایانه‌ها قادرند دستورالعمل‌ها را با سرعت و دقت بسیار بالا اجرا کنند و چرا طراحی بهینه آن نقش حیاتی در عملکرد کلی سیستم دارد.

در قلب هر پردازنده، واحدهایی وجود دارند که مسئول تحلیل، کنترل، ذخیره و اجرای دستورالعمل‌ها هستند. یکی از مهم‌ترین بخش‌ها، واحد کنترل (Control Unit) است. این بخش وظیفه دارد تا دستورالعمل‌های موجود در حافظه را خوانده، تفسیر کرده و سیگنال‌های کنترلی لازم را برای اجرای آن‌ها توسط دیگر اجزای پردازنده صادر کند. برای نمونه، هنگام اجرای یک عملیات جمع ساده، واحد کنترل مشخص می‌کند که کدام داده‌ها باید از حافظه فراخوانی شوند، به کدام رجیستر منتقل شوند، و چه نوع عملیاتی روی آن‌ها اعمال گردد.

در کنار واحد کنترل، واحد حساب و منطق (Arithmetic Logic Unit - ALU) قرار دارد که وظیفه انجام محاسبات عددی و منطقی را بر عهده دارد. این واحد قادر است اعمالی نظیر جمع، تفریق، ضرب، تقسیم، و همچنین عملیات منطقی مانند AND، OR، XOR و NOT را انجام دهد. در پردازنده‌های پیشرفته‌تر، معمولاً یک یا چند واحد ALU وجود دارد که به‌صورت موازی عمل می‌کنند تا سرعت اجرای دستورات افزایش یابد. همچنین در بسیاری از پردازنده‌ها یک واحد خاص به‌نام FPU (واحد محاسبات اعشاری) نیز وجود دارد که برای انجام محاسبات با اعداد اعشاری طراحی شده است.

بخش مهم دیگر در معماری CPU، رجیسترها هستند. رجیسترها حافظه‌های بسیار کوچک، سریع و داخلی هستند که برای نگهداری داده‌ها و نتایج موقت در طول اجرای برنامه استفاده می‌شوند. هر پردازنده مجموعه‌ای از رجیسترهای عمومی (General-purpose Registers) و رجیسترهای خاص (Special-purpose Registers) دارد. رجیستر شمارنده برنامه (Program Counter - PC) یکی از رجیسترهای حیاتی است که آدرس دستورالعمل بعدی را در خود نگه می‌دارد. رجیسترهایی نظیر Stack Pointer، Instruction Register و Status Register نیز نقش کلیدی در اجرای دقیق برنامه‌ها دارند.

یکی از ویژگی‌های مهم پردازنده‌های مدرن، بهره‌گیری از حافظه کش (Cache Memory) است. کش حافظه‌ای بسیار سریع ولی با ظرفیت محدود است که بین CPU و حافظه اصلی قرار می‌گیرد. داده‌ها و دستورالعمل‌هایی که به‌صورت مکرر استفاده می‌شوند، در کش نگهداری می‌شوند تا CPU بتواند با سرعت بسیار بالا به آن‌ها دسترسی داشته باشد. معمولاً کش در سه سطح L1، L2 و L3 طبقه‌بندی می‌شود که L1 سریع‌ترین و کوچک‌ترین، و L3 کندتر ولی بزرگ‌تر است. استفاده هوشمندانه از کش باعث افزایش چشم‌گیر کارایی سیستم می‌شود.

در معماری پردازنده‌های مدرن، استفاده از پایپ‌لاینینگ (Pipelining) نیز بسیار رایج است. پایپ‌لاینینگ به معنای شکستن اجرای هر دستورالعمل به مراحل کوچکتر و اجرای هم‌زمان این مراحل برای چند دستورالعمل است. برای مثال، اگر یک دستورالعمل شامل مراحل واکشی (Fetch)، رمزگشایی (Decode)، اجرا (Execute)، و نوشتن نتیجه (Write Back) باشد، پردازنده می‌تواند هر یک از این مراحل را به‌صورت موازی برای دستورالعمل‌های مختلف انجام دهد. این تکنیک باعث می‌شود که در هر چرخه ساعت، چندین دستورالعمل در حال اجرا باشند، و در نتیجه بهره‌وری CPU به طرز چشم‌گیری افزایش یابد.

یکی دیگر از مفاهیم کلیدی در طراحی پردازنده، اجرای هم‌زمان چند رشته (Multithreading) و چند هسته‌ای بودن (Multi-core) است. در معماری‌های مدرن، یک CPU ممکن است شامل چندین هسته مستقل باشد که هرکدام قادر به اجرای برنامه‌ای جداگانه هستند. این ساختار باعث می‌شود رایانه‌ها بتوانند به‌صورت واقعی چندین برنامه را به‌طور هم‌زمان اجرا کنند. همچنین تکنیک‌هایی مانند Hyper-Threading در پردازنده‌های Intel باعث می‌شود هر هسته بتواند دو thread را به‌صورت موازی مدیریت کند و در نتیجه کارایی بهتری ارائه دهد.

مسیرهای داده (Data Paths) نیز بخشی از معماری داخلی CPU هستند که مسئول جابجایی داده‌ها بین اجزای مختلف پردازنده‌اند. این مسیرها به کمک گذرگاه‌ها (Buses) طراحی شده‌اند و معمولاً شامل گذرگاه‌های آدرس، داده و کنترل هستند. طراحی بهینه مسیرهای داده باعث کاهش تأخیر در دسترسی به داده‌ها و افزایش سرعت پردازش می‌شود.

علاوه بر این‌ها، در پردازنده‌های امروزی واحدهایی برای پیش‌بینی شاخه (Branch Prediction)، اجرای حدسی (Speculative Execution) و مرتب‌سازی مجدد دستورالعمل‌ها (Out-of-Order Execution) نیز وجود دارد. این قابلیت‌ها باعث می‌شوند که پردازنده بتواند دستورالعمل‌ها را با بازدهی بیشتری اجرا کند، حتی در شرایطی که برنامه شامل دستورات شرطی و وابستگی‌های زیاد بین داده‌ها باشد. برای مثال، در اجرای حدسی، پردازنده با استفاده از الگوریتم‌های هوشمند مسیر محتمل اجرای برنامه را حدس می‌زند و ادامه دستورات را بدون وقفه اجرا می‌کند؛ اگر حدس اشتباه باشد، نتیجه اجرای نادرست کنار گذاشته می‌شود.

معماری داخلی CPU ترکیبی پیچیده و منسجم از واحدهای پردازشی، مسیرهای داده، سیستم‌های کنترلی و حافظه‌های سریع است که همگی با هدف افزایش سرعت، دقت، و بازدهی طراحی شده‌اند. این معماری همواره در حال تحول است و هر نسل جدید از پردازنده‌ها با بهینه‌سازی‌های بیشتر، قابلیت‌های پیشرفته‌تر، و توان محاسباتی بالاتر به بازار عرضه می‌شوند تا پاسخگوی نیازهای روزافزون محاسباتی در عصر دیجیتال باشند.

معماری داخلی پردازنده گرافیکی (GPU) از جمله پیچیده‌ترین و در عین حال تحول‌برانگیزترین ساختارهای سخت‌افزاری در دنیای محاسبات مدرن است. برخلاف پردازنده مرکزی (CPU) که برای اجرای مجموعه‌ای از دستورالعمل‌های عمومی به‌صورت ترتیبی طراحی شده، GPU با هدف انجام هم‌زمان هزاران عملیات مشابه بر روی داده‌های گسترده، به شکلی بسیار موازی سازمان‌دهی شده است. این ویژگی، GPU را به ابزاری بی‌نظیر برای کاربردهایی چون رندرینگ گرافیکی، یادگیری ماشین، شبیه‌سازی‌های علمی، تحلیل کلان‌داده‌ها، و استخراج رمز ارزها تبدیل کرده است.

در قلب معماری GPU، واحدهای کوچکتر و متعدد پردازشی وجود دارند که تحت عنوان "هسته‌های CUDA" در پردازنده‌های NVIDIA و "Stream Processors" در AMD شناخته می‌شوند. این هسته‌ها بسیار ساده‌تر از هسته‌های CPU هستند اما در تعداد بالا (صدها تا هزاران هسته) درون یک GPU جای می‌گیرند. معماری GPU به گونه‌ای طراحی شده که این هسته‌ها بتوانند به صورت هم‌زمان عملیات مشابهی را روی مجموعه‌های بزرگ داده انجام دهند. این ساختار باعث می‌شود تا GPU در پردازش‌های برداری یا ماتریسی بسیار مؤثرتر از CPU عمل کند.

در GPUهای مدرن، این هسته‌ها در بلوک‌هایی به نام "چندپردازنده‌های جریانی" (Streaming Multiprocessors - SM) در NVIDIA یا "Compute Units" در AMD سازماندهی شده‌اند. هر SM دارای تعدادی هسته، رجیستر، حافظه اشتراکی (Shared Memory) و برنامه‌ریزهای داخلی است. برنامه‌ریز یا Scheduler وظیفه تخصیص threadها به هسته‌ها و زمان‌بندی اجرای آن‌ها را بر عهده دارد. این ساختار امکان اجرای هزاران thread به صورت هم‌زمان را فراهم می‌آورد، به‌ویژه در الگوریتم‌هایی که حجم بالایی از داده را به شکل مستقل باید پردازش کنند، مانند فیلترهای تصویری، تحلیل صوت یا آموزش شبکه‌های عصبی.

واحدهای پردازش گرافیکی همچنین شامل ساختارهای کنترلی پیچیده‌ای هستند که برای مدیریت threadها و بلوک‌های thread طراحی شده‌اند. در چارچوب CUDA (محیط توسعه NVIDIA)، threadها درون بلوک‌ها (Blocks) و بلوک‌ها درون شبکه‌ها (Grids) سازمان‌دهی می‌شوند. هر بلوک می‌تواند از حافظه اشتراکی برای همکاری بین threadها استفاده کند. این سازمان‌دهی سلسله‌مراتبی به GPU امکان می‌دهد تا مقیاس‌پذیری افقی عظیمی داشته باشد و برای کار روی داده‌های بسیار بزرگ بدون افت عملکرد قابل توجه، به راحتی تقسیم وظایف انجام دهد.

در کنار هسته‌های پردازشی، GPUها دارای واحدهای اختصاصی برای انجام وظایف خاص هستند، از جمله واحدهای رسترایزر، بافرهای فریم، واحدهای تکسچر و واحدهای ALU ویژه برای عملیات برداری یا ماتریسی. این اجزا برای شتاب‌دهی به عملیات گرافیکی طراحی شده‌اند، ولی در سال‌های اخیر به کمک ابزارهایی مانند CUDA، OpenCL و Vulkan، در کاربردهای عمومی محاسباتی GPGPU (General Purpose computing on GPU) نیز به‌کار گرفته می‌شوند.

حافظه در معماری GPU نقش حیاتی دارد و ساختار آن با CPU تفاوت چشم‌گیری دارد. در GPU معمولاً انواع مختلفی از حافظه وجود دارد: حافظه جهانی (Global Memory) که بین تمام هسته‌ها مشترک است، حافظه اشتراکی (Shared Memory) که به‌طور محلی در اختیار هر بلوک است، و حافظه خصوصی یا رجیسترها که به هر thread اختصاص دارد. بهره‌برداری مؤثر از این سلسله‌مراتب حافظه تأثیر مستقیمی بر عملکرد الگوریتم دارد، چرا که دسترسی به حافظه جهانی بسیار کندتر از حافظه‌های محلی یا رجیستری است.

از لحاظ زمان‌بندی و اجرای دستورالعمل، GPUها برخلاف CPU معمولاً فاقد ویژگی‌هایی مانند پیش‌بینی شاخه (Branch Prediction) یا اجرای خارج از ترتیب (Out-of-Order Execution) هستند. دلیل این موضوع این است که طراحی GPU بر اساس پردازش هم‌زمان و تکراری تعداد زیادی thread با رفتار یکسان است. بنابراین، پیچیدگی‌های مربوط به دستورهای شرطی یا وابستگی‌های شدید داده‌ای در GPU بسیار کمتر از CPU است و در صورت بروز چنین شرایطی، عملکرد GPU ممکن است افت کند.

GPUهای مدرن اغلب شامل رابط‌هایی برای ارتباط با حافظه اصلی سیستم یا سایر پردازنده‌ها نیز هستند، که از طریق گذرگاه‌های پرسرعت مانند PCI Express یا NVLink انجام می‌شود. سرعت و پهنای باند این ارتباط، به‌ویژه در کاربردهای علمی و یادگیری ماشین که داده‌های حجیم در جریان هستند، اهمیت بسیار بالایی دارد.

در نهایت، معماری GPU از ابتدا با هدف موازی‌سازی گسترده طراحی شده و در گذر زمان با افزودن قابلیت‌های جدید، به نقطه‌ای رسیده که امروزه نه تنها در پردازش گرافیکی بلکه در بسیاری از حوزه‌های محاسباتی سنگین، جایگاه ویژه‌ای یافته است. تفاوت بنیادین آن با CPU در میزان موازی‌سازی، ساختار

ساده‌تر هسته‌ها، حافظه‌های چندسطحی تخصصی، و زمان‌بندی برای اجرای انبوهی از threadها است. درک معماری داخلی GPU نه تنها برای توسعه بازی‌های رایانه‌ای و گرافیک سه‌بعدی بلکه برای پژوهش‌های علمی، رمزنگاری، یادگیری عمیق و تحلیل داده‌های بزرگ ضرورتی انکارناپذیر است.

واحدهای پردازشی در GPU

واحدهای پردازشی GPU اساساً هسته‌های تخصصی هستند که به دقت برای اجرای موازی مهندسی شده‌اند. برخلاف هسته‌های CPU سنتی که معمولاً تعداد کمی از وظایف پیچیده را به صورت متوالی انجام می‌دهند، هسته‌های GPU (مانند هسته‌های CUDA انویدیا یا پردازنده‌های جریانی/واحدهای محاسباتی AMD) برای تجزیه مسائل محاسباتی بزرگ به هزاران قطعه کوچکتر و قابل مدیریت طراحی شده‌اند که سپس به صورت همزمان حل می‌شوند. این پارادایم محاسبات موازی به طور استثنایی برای حجم کاری‌های پرتقاضا که در آن عملیات یکسان باید در مجموعه‌های داده گسترده اعمال شود، مناسب است.

در یک مقایسه مستقیم، هسته‌های CPU برای "سرعت هر هسته، تأخیر و تنوع دستورالعمل" بهینه‌سازی شده‌اند، در حالی که هسته‌های CUDA (نماینده GPUها) بر "توان عملیاتی عظیم، موازی‌سازی وظیفه و چگالی رشته" تمرکز دارند. این مقایسه یک مبادله اساسی در معماری پردازنده را برجسته می‌کند. CPUها به دقت برای وظایف متوالی با تأخیر کم بهینه‌سازی شده‌اند که برای پاسخگویی کلی سیستم و عملکرد برنامه‌های تک‌رشته‌ای حیاتی است. GPUها، با طراحی خود، برخی از انعطاف‌پذیری و تأخیر هر هسته را به نفع دستیابی به توان عملیاتی عظیم برای وظایف بسیار موازی‌شونده قربانی می‌کنند. این مبادله ذاتی توضیح می‌دهد که چرا GPUها به طور جهانی جایگزین CPUها نمی‌شوند، بلکه به عنوان مکمل‌های قدرتمند عمل می‌کنند و در حوزه‌های محاسباتی خاصی برتری دارند. این تمایز معماری مستقیماً بخش‌بندی بازار CPU و GPU را شکل می‌دهد. CPUها نقش اصلی خود را برای سیستم‌عامل‌ها، برنامه‌های عمومی و وظایفی که نیاز به اجرای سریع تک‌رشته‌ای دارند، حفظ می‌کنند. در مقابل، GPUها بر حوزه‌هایی مانند رندرینگ گرافیک، هوش مصنوعی و محاسبات علمی تسلط دارند، جایی که موازی‌سازی داده‌های عظیم و توان عملیاتی بالا از اهمیت بالایی برخوردار است. این نشان‌دهنده چشم‌انداز محاسباتی آینده است که با سخت‌افزارهای به‌طور فزاینده تخصصی مشخص می‌شود، جایی که دستیابی به عملکرد بهینه برای یک حجم کاری معین، انتخاب مناسب‌ترین واحد پردازشی را ضروری می‌سازد، نه تکیه بر یک راه‌حل "یک‌اندازه برای همه".

۱. هسته های CUDA (CUDA Cores)

هسته های CUDA پردازنده‌های عمومی بنیادی هستند که در GPUهای NVIDIA ادغام شده‌اند و به طور خاص برای محاسبات موازی مهندسی شده‌اند. اصل عملیاتی آن‌ها شامل تجزیه مسائل محاسباتی بزرگ

و پیچیده به هزاران وظیفه کوچکتر و مستقل است که سپس به صورت همزمان پردازش می‌شوند. هر هسته CUDA قادر به اجرای یک عملیات در هر چرخه ساعت است. این هسته‌ها بسیار انعطاف‌پذیر هستند و برای پاسخگویی به طیف گسترده‌ای از نیازهای محاسباتی عمومی، از شبیه‌سازی‌های علمی پیچیده گرفته تا فرآیندهای رندرینگ پیشرفته و الگوریتم‌های یادگیری ماشین سنتی، ساخته شده‌اند.

هسته‌های CUDA به طور بهینه برای حجم کاری‌های پرتقاضا، از جمله یادگیری ماشین، تحلیل‌های بلادرنگ و برنامه‌های مختلف نرم‌افزار به عنوان سرویس (SaaS) مناسب هستند. در حوزه پلتفرم‌های تحلیلی، آن‌ها به طور قابل توجهی خطوط لوله ETL (استخراج، تبدیل، بارگذاری)، پرس‌وجوهای بلادرنگ و پردازش ستونی را بر روی مجموعه‌های داده عظیم تسریع می‌کنند. برای برنامه‌هایی که شامل ویرایش ویدئو، رمزگذاری، پخش جریانی و جلوه‌های پس‌تولید هستند، هسته‌های CUDA پردازش همزمان چندین فریم ویدئو و لایه‌های رندرینگ را تسهیل می‌کنند و عملکرد بلادرنگ را امکان‌پذیر می‌سازند. علاوه بر این، در برنامه‌های SaaS تصمیم‌گیری بلادرنگ، مانند تشخیص تقلب یا نگهداری پیش‌بینی‌کننده، آن‌ها اجرای موازی مدل‌های پیچیده و منطق مبتنی بر قانون را امکان‌پذیر می‌سازند و امکان پاسخ‌های تقریباً فوری را فراهم می‌کنند.

در سال ۲۰۰۷، انویدیا CUDA را منتشر کرد، یک لایه نرم‌افزاری که پردازش موازی را در GPU در دسترس قرار داد. این توسعه، برنامه‌نویسی GPU را برای مخاطبان وسیع‌تری باز کرد و "محاسبات GPU را بسیار رایج‌تر" کرد. این موضوع فراتر از یک انتشار نرم‌افزاری ساده بود؛ این یک حرکت استراتژیک بود که پتانسیل محاسباتی عمومی GPUها را به طور اساسی باز کرد و محبوبیت بخشید. این تحول عامل مهمی در تسلط قابل توجه انویدیا بر بازار، به ویژه در حوزه‌های رو به رشد هوش مصنوعی/یادگیری ماشین بوده است. این موضوع به وضوح نشان می‌دهد که نوآوری سخت‌افزاری، در حالی که ضروری است، اغلب به تنهایی برای پذیرش گسترده و باز کردن پتانسیل کامل فناوری‌های جدید کافی نیست. یک اکوسیستم نرم‌افزاری قوی، قابل دسترس و با پشتیبانی جامع، به همان اندازه، اگر نگوییم بیشتر، حیاتی است. موفقیت انویدیا با CUDA به عنوان یک مطالعه موردی قانع‌کننده عمل می‌کند که نشان می‌دهد چگونه یک استراتژی نرم‌افزاری با اجرای خوب می‌تواند کاربرد سخت‌افزار را هدایت کند، یک جامعه توسعه‌دهنده پر رونق را پرورش دهد و در نهایت رهبری بازار را تضمین کند.

۲. هسته های Tensor (Tensor Cores)

تعریف و بهینه‌سازی برای عملیات ماتریسی

هسته های Tensor واحدهای پردازشی بسیار تخصصی هستند که در GPUهای NVIDIA تعبیه شده‌اند و به دقت برای تسریع وظایف یادگیری عمیق، از جمله آموزش مدل و استنتاج، مهندسی شده‌اند. آن‌ها به طور خاص برای محاسبات ماتریسی بهینه‌سازی شده‌اند و برای پردازش عملیات ماتریسی در مقیاس

بزرگ در یک انفجار واحد و بسیار کارآمد طراحی شده‌اند. مزیت اصلی معماری آن‌ها نسبت به هسته‌های CUDA استاندارد در قابلیت آن‌ها برای انجام عملیات ماتریسی با دقت ترکیبی در توان عملیاتی به طور قابل توجهی بالاتر است، که یک عامل حیاتی برای عملکرد یادگیری عمیق است.

هسته‌های Tensor نقش محوری در افزایش کارایی شبکه‌های عصبی مدرن ایفا می‌کنند و آموزش و استقرار مدل‌های ترانسفورمر با میلیاردها پارامتر و سیستم‌های هوش مصنوعی در مقیاس بزرگ را امکان‌پذیر می‌سازند. آن‌ها در کاهش چشمگیر زمان آموزش برای مدل‌های پیچیده هوش مصنوعی و بهبود قابل توجه عملکرد استنتاج، که برای برنامه‌های هوش مصنوعی بلادرنگ مانند سیستم‌های توصیه‌گر، وسایل نقلیه خودران و سیستم‌های تشخیص صدا حیاتی است، نقش مهمی دارند.

تکامل و دقت‌های محاسباتی (FP16, BF16, INT8, FP8)

هسته‌های Tensor با معماری Volta انویدیا در سال ۲۰۱۷ معرفی شدند و در ابتدا کاربردهای مرکز داده را هدف قرار دادند. نسل اول (Volta) عمدتاً از فرمت‌های FP16 (نقطه شناور با دقت نیمه) استفاده می‌کرد که افزایش قابل توجهی در عملکرد محاسباتی ایجاد کرد. نسل‌های بعدی شاهد پیشرفت‌های چشمگیری بودند: هسته‌های Tensor Turing پشتیبانی را برای فرمت‌های با دقت کمتر مانند INT8، INT4 و دقت ۱ بیتی باینری گسترش دادند. هسته‌های Tensor Ampere با افزودن پشتیبانی از انواع داده TF32 و BF32، قابلیت‌ها را بیشتر گسترش دادند. اخیراً، معماری Hopper (که با GPU H100 نمونه‌سازی شده است) فرمت‌های دقت FP8 را معرفی کرد که انویدیا ادعا می‌کند می‌تواند عملکرد مدل‌های زبان بزرگ را تا ۳۰ برابر نسبت به نسل قبلی تسریع کند. این تکامل مداوم در دقت‌های محاسباتی پشتیبانی‌شده، برای فعال کردن مدل‌های یادگیری عمیق سریع‌تر و کارآمدتر از نظر انرژی از طریق تکنیک‌های آموزش با دقت ترکیبی، محوری است.

هسته‌های Tensor با استفاده از "فرمت‌های با دقت کمتر مانند FP16، BF16، INT8 و FP8" یا از طریق "حساب با دقت ترکیبی" به سرعت برتر خود دست می‌یابند. در مقابل، هسته‌های CUDA معمولاً با دقت بالاتر (FP32/FP64) برای وظایفی که "دقت نمی‌تواند به خطر بیفتد" عمل می‌کنند. این تمایز یک مبادله مهندسی حیاتی را آشکار می‌کند: برای بخش قابل توجهی از وظایف هوش مصنوعی/یادگیری ماشین، کاهش جزئی در دقت عددی (به عنوان مثال، از FP32 به FP16 یا INT8) منجر به افزایش نامتناسب و بزرگی در عملکرد با حداقل، و اغلب قابل قبول، از دست دادن دقت مدل می‌شود. این درک اساسی مستقیماً توسعه واحدهای سخت‌افزاری تخصصی مانند هسته‌های Tensor را تحریک کرد. این روند به شدت نشان می‌دهد که شتاب‌دهنده‌های هوش مصنوعی آینده به کاوش و اتخاذ فرمت‌های دقت حتی پایین‌تر و نمایش‌های عددی تخصصی جدید برای به حداکثر رساندن توان محاسباتی و کارایی انرژی ادامه خواهند داد. این می‌تواند به طور بالقوه منجر به توسعه رویکردهای الگوریتمی جدیدی شود که ذاتاً نسبت به کاهش دقت

تحمل بیشتری دارند. علاوه بر این، این امر نشان می‌دهد که چارچوب‌ها و کتابخانه‌های نرم‌افزاری باید به طور مداوم تکامل یابند تا این قابلیت‌های دقت ترکیبی به‌طور فزاینده متنوع را به طور مؤثر مدیریت، استفاده و بهینه‌سازی کنند.

معرفی و تکامل مداوم هسته‌های Tensor در معماری‌های انویدیا (Volta، Turing، Ampere، Hopper) مستقیماً با افزایش تقاضاها و الگوهای محاسباتی خاص مشاهده شده در حجم کاری‌های یادگیری عمیق مرتبط است. در ابتدا، GPUها به عنوان پردازنده‌های موازی عمومی عمل می‌کردند که عمدتاً توسط هسته‌های CUDA تعریف می‌شدند. با این حال، با افزایش اهمیت حجم کاری‌های هوش مصنوعی و روشن شدن نیازهای محاسباتی آن‌ها (به ویژه شیوع ضرب ماتریس)، ضرورت واحدهای سخت‌افزاری بسیار کارآمد و اختصاصی مستقیماً منجر به ایجاد و اصلاح هسته‌های Tensor شد. این پیشرفت به عنوان یک نمونه واضح و قانع‌کننده از تخصصی‌سازی سخت‌افزاری مبتنی بر حجم کاری عمل می‌کند. این الگو به شدت نشان می‌دهد که با ظهور پارادایم‌های محاسباتی غالب جدید (مانند محاسبات کوانتومی پیشرفته، معماری‌های محاسباتی نورومورفیک جدید)، صنعت احتمالاً شاهد موج‌های بیشتری از تخصصی‌سازی سخت‌افزاری فراتر از طراحی‌های GPU فعلی خواهد بود. این امر این ایده را تقویت می‌کند که GPU "عمومی" به تدریج به یک شتاب‌دهنده "ناهمگن" تبدیل می‌شود که مجموعه‌ای متنوع از واحدهای تخصصی را در خود جای داده است که هر کدام برای یک وظیفه محاسباتی خاص بهینه‌سازی شده‌اند.

۳. هسته‌های RT (Ray Tracing Cores)

هسته‌های RT واحدهای سخت‌افزاری تخصصی هستند که توسط NVIDIA معرفی شده‌اند، که با معماری Turing آغاز شد و به طور خاص برای تسریع ردیابی پرتو در زمان واقعی مهندسی شده‌اند. ردیابی پرتو یک تکنیک رندرینگ پیچیده است که با شبیه‌سازی مسیر فیزیکی پرتوهای نور و تعاملات پیچیده آن‌ها با اشیاء مجازی در یک صحنه، تصاویر را تولید می‌کند. این امکان ایجاد جلوه‌های بسیار واقع‌گرایانه مانند سایه‌های نرم و دقیق و بازتاب‌های واقعی را فراهم می‌کند. هسته‌های RT به طور خاص برای تسریع عملیات ریاضی فشرده محاسباتی مورد نیاز برای ردیابی پرتو کارآمد، به ویژه پیمایش سلسله‌مراتب حجم محدود (BVH)، طراحی شده‌اند.

هسته‌های RT جزء اصلی فناوری RTX انویدیا هستند که تولید تصاویر تعاملی را تسهیل می‌کند که به طور پویا به نورپردازی، سایه‌ها و بازتاب‌ها در زمان واقعی پاسخ می‌دهند. آن‌ها در ارائه تجربه‌های بصری فراگیرتر و تعاملی‌تر در حوزه‌های مختلف، از جمله بازی، تجسم معماری و طراحی محصول، نقش اساسی دارند. علاوه بر این، چندپردازنده‌های جریانی Ampere (SMs) به گونه‌ای طراحی شده‌اند که امکان اجرای

همزمان حجم کاری هسته RT و هسته CUDA را فراهم می‌کنند و کارایی کلی رندرینگ، به ویژه برای صحنه‌های پیچیده، را به طور قابل توجهی افزایش می‌دهند.

هر دو بر یک رویکرد عملی حیاتی تأکید می‌کنند: "از آنجایی که ردیابی پرتو هنوز از نظر محاسباتی فشرده است، بسیاری از توسعه‌دهندگان رویکرد رندرینگ ترکیبی را انتخاب می‌کنند که در آن برخی از جلوه‌های گرافیکی، مانند سایه‌ها و بازتاب‌ها، با استفاده از ردیابی پرتو انجام می‌شوند، در حالی که صحنه باقی‌مانده با استفاده از رسترایزیشن با عملکرد بالاتر رندر می‌شود." این بیان یک راه‌حل مهندسی استراتژیک برای هزینه محاسباتی بالای ردیابی پرتو کامل و بلادرنگ را آشکار می‌کند. به جای یک پیاده‌سازی همه‌جانبه، GPUها سخت‌افزار تخصصی (هسته‌های RT) را برای تسریع پرتقاضاترین جنبه‌های ردیابی پرتو ادغام می‌کنند، در حالی که به طور همزمان از روش‌های سنتی و کارآمدتر (رسترایزیشن) برای سایر بخش‌های خط لوله رندرینگ استفاده می‌کنند. این رویکرد متعادل امکان ترکیبی قانع‌کننده از وفاداری بصری پیشرفته و عملکرد بلادرنگ قابل قبول را فراهم می‌کند. این رویکرد رندرینگ "ترکیبی" به احتمال زیاد ادامه خواهد یافت و تکامل خواهد یافت، زیرا پذیرش تدریجی تکنیک‌های رندرینگ جدید و از نظر محاسباتی فشرده را بدون نیاز به قربانی کردن کامل عملکرد کلی تسهیل می‌کند. این امر نشان می‌دهد که خطوط لوله گرافیکی آینده همچنان ترکیبی پیچیده از سخت‌افزار تخصصی برای جلوه‌های بصری پیشرفته و واحدهای عمومی بسیار بهینه‌سازی شده برای وظایف رندرینگ بنیادی باقی خواهند ماند.

واحد های حافظه در GPU

واحد حافظه در پردازنده‌های گرافیکی (GPU) یکی از حیاتی‌ترین و در عین حال پیچیده‌ترین بخش‌های این معماری است. دلیل اهمیت این موضوع در آن است که عملکرد نهایی یک GPU در بسیاری از کاربردهای محاسباتی (اعم از رندرینگ گرافیکی یا تحلیل داده‌های علمی) به شدت به ساختار و نحوه مدیریت حافظه وابسته است. برخلاف CPU که دارای سلسله‌مراتب ساده‌تری از حافظه است، GPU به‌منظور پشتیبانی از هزاران thread هم‌زمان، از ساختار حافظه‌ای چندلایه، چندسطحی و با ویژگی‌های خاص بهره می‌برد که درک آن برای طراحی الگوریتم‌های کارآمد روی GPU ضروری است.

در معماری GPU، انواع مختلفی از حافظه وجود دارند که هرکدام ویژگی‌ها، ظرفیت، سرعت و حوزه دسترسی خاص خود را دارند. مهم‌ترین این حافظه‌ها عبارتند از: حافظه جهانی (Global Memory)، حافظه اشتراکی (Shared Memory)، حافظه خصوصی یا رجیستری (Private/Register Memory)، حافظه محلی (Local Memory)، حافظه ثابت (Constant Memory)، و حافظه متنی (Texture Memory). هرکدام از این سطوح نقش مشخصی در سازماندهی داده‌ها در حین اجرای threadهای موازی ایفا می‌کنند.

حافظه جهانی (Global Memory) اصلی‌ترین و بزرگ‌ترین بخش حافظه در GPU است که تمامی threadها در تمام بلوک‌ها (Blocks) به آن دسترسی دارند. این حافظه در خارج از واحدهای محاسباتی قرار دارد و معمولاً روی حافظه‌های GDDR یا HBM مستقر است. دسترسی به این حافظه نسبتاً کند است (در مقایسه با حافظه‌های داخلی‌تر) و به همین دلیل استفاده بهینه از آن در الگوریتم‌ها نقش بسیار مهمی در افزایش کارایی دارد. یکی از نکات کلیدی در استفاده از Global Memory، هم‌ترازسازی دسترسی‌ها (Memory Coalescing) است؛ یعنی اگر threadها به مکان‌های متوالی از حافظه دسترسی داشته باشند، GPU می‌تواند این دسترسی‌ها را در یک عملیات ترکیب کرده و پهنای باند حافظه را بهینه کند.

حافظه اشتراکی (Shared Memory) یکی از سریع‌ترین و ارزشمندترین انواع حافظه در GPU است که در داخل هر چندپردازنده‌ی جریانی (SM یا Compute Unit) قرار دارد. این حافظه به‌صورت محلی و فقط در اختیار threadهای یک Block مشخص است. ویژگی مهم Shared Memory در آن است که سرعت بسیار بالایی دارد و برای پیاده‌سازی همکاری بین threadها در داخل یک بلوک بسیار مؤثر است. برای مثال، در الگوریتم‌های ماتریسی یا تحلیل سیگنال، داده‌ها ابتدا از حافظه جهانی به حافظه اشتراکی منتقل شده، سپس پردازش انجام می‌شود و در پایان، نتایج به Global Memory بازگردانده می‌شود. این تکنیک باعث کاهش چشمگیر زمان اجرا می‌شود.

رجیسترها (Register Memory) سریع‌ترین نوع حافظه در GPU هستند که به هر thread اختصاص داده می‌شوند و برای نگهداری داده‌های موقتی و متغیرهای محلی استفاده می‌شوند. استفاده از رجیسترها زمانی مؤثر است که داده‌ها فقط توسط یک thread قابل دسترسی باشند. تعداد رجیسترهای موجود محدود است و اگر برنامه‌نویس یا کامپایلر از آن‌ها بیش از حد استفاده کند، داده‌ها به حافظه محلی (Local Memory) منتقل می‌شوند که کندتر است و در واقع بر روی حافظه جهانی شبیه‌سازی می‌شود.

حافظه محلی (Local Memory) برخلاف نامش، یک حافظه محلی واقعی نیست بلکه بخشی از Global Memory است که در صورت کمبود رجیسترها به‌عنوان جایگزین استفاده می‌شود. این حافظه برای داده‌هایی است که فقط برای یک thread خاص قابل دسترسی است ولی به دلیل مکان فیزیکی‌اش، سرعت پایینی دارد. لذا طراحی به‌گونه‌ای که استفاده از رجیسترها را به حداکثر برساند و نیاز به Local Memory را کاهش دهد، یکی از اصول بهینه‌سازی در برنامه‌نویسی GPU است.

حافظه ثابت (Constant Memory) حافظه‌ای فقط‌خواندنی است که برای ذخیره‌سازی داده‌هایی که در طول اجرای Kernel تغییر نمی‌کنند به‌کار می‌رود. این حافظه برای همه threadها قابل دسترسی است و وقتی همه threadها به یک مکان از Constant Memory دسترسی داشته باشند، می‌تواند عملکرد بسیار خوبی داشته باشد زیرا توسط cache داخلی آن شتاب داده می‌شود.

حافظه متنی (Texture Memory) و حافظه سطحی (Surface Memory) نیز به‌طور خاص برای پردازش‌های گرافیکی طراحی شده‌اند اما در برنامه‌نویسی عمومی GPU (GPGPU) نیز قابل استفاده‌اند.

حافظه متنی برای داده‌هایی با الگوی دسترسی فضایی خاص، مانند تصاویر یا ماتریس‌های بزرگ دوبعدی استفاده می‌شود و دارای قابلیت‌هایی چون فیلترگذاری و درونیابی (interpolation) است. این حافظه‌ها توسط cacheهای مخصوص پشتیبانی می‌شوند و می‌توانند در بعضی الگوریتم‌ها مزیت‌های عملکردی قابل توجهی ایجاد کنند.

به‌طور کلی، طراحی معماری حافظه در GPU برای پشتیبانی از پردازش‌های massively parallel انجام شده است، به این معنا که صدها یا هزاران thread باید بتوانند هم‌زمان و با تأخیر کم به داده‌های مورد نیاز خود دسترسی پیدا کنند. این نیاز منجر به طراحی سلسله‌مراتبی پیچیده‌ای شده که هر سطح حافظه مزایا و معایب خاص خود را دارد. بهره‌برداری مؤثر از این ساختار نیازمند درک عمیق از رفتار threadها، الگوی دسترسی داده و تقسیم بهینه وظایف در سطح بلوک و شبکه است.

در نهایت، واحد حافظه در GPU نه تنها به‌عنوان زیرساختی برای ذخیره‌سازی داده‌ها عمل می‌کند، بلکه بخش مهمی از هوش معماری GPU محسوب می‌شود. برنامه‌نویس موفق در زمینه CUDA یا OpenCL کسی است که بتواند این ساختار حافظه‌ای را به‌خوبی درک کرده و الگوریتم‌هایی بنویسد که از حافظه‌های سریع‌تر بیشتر استفاده کنند، دسترسی‌های پراکنده را کاهش دهند، و الگوهای همکاری بین threadها را بهینه‌سازی نمایند. این اصول کلیدی در رسیدن به حداکثر توان محاسباتی GPU نقش تعیین‌کننده دارند.

بررسی نسل های مختلف GPU و پیشرفت آن ها

پیش از ظهور رسمی GPU، سخت‌افزارهای گرافیکی برای کاربردهای خاصی مانند CAD، شبیه‌سازی و بازی توسعه یافتند. در این دوره (دهه‌های ۱۹۷۰ تا ۱۹۹۹)، بهبود عملکرد سخت‌افزارهای گرافیکی با نرخ تقریباً ۲.۵ برابر در سال، از سرعت پیش‌بینی‌شده قانون مور پیشی گرفت. این رشد سریع به دلیل بهره‌برداری مؤثر طراحان سخت‌افزار از ماهیت موازی محاسبات گرافیک کامپیوتری بود.

در ابتدا، تولیدکنندگان سخت‌افزار از رابط‌های اختصاصی خود استفاده می‌کردند که منجر به "جنگ APIها" و تکه‌تکه شدن بازار شد. این عدم ثبات، رشد صنعت را به خطر می‌انداخت. اما با معرفی DirectX مایکروسافت، استانداردسازی در زمینه گرافیک آغاز شد و این چارچوب به تعیین‌کننده دوره‌های مختلف GPU تبدیل گشت. DirectX 7.0 در سپتامبر ۱۹۹۹ عرضه شد و پیشرفت قابل توجهی در قابلیت‌های پردازش گرافیک ایجاد کرد. این نسخه، شتاب‌دهی سخت‌افزاری برای عملیات تبدیل و نورپردازی (T&L) و قابلیت تخصیص بافرهای رأس مستقیماً در حافظه سخت‌افزار را معرفی کرد. این پیشرفت‌ها، Direct3D مایکروسافت را نسبت به استاندارد رقیب OpenGL برتری بخشید.

نقطه عطف اصلی در سال ۱۹۹۹ با معرفی GeForce 256 توسط NVIDIA رقم خورد، که این شرکت آن را به عنوان "اولین GPU جهان" معرفی کرد. این پردازنده تک‌تراشه‌ای، موتورهای تبدیل و نورپردازی (T&L)، تنظیم/برش مثلث و رندرینگ را یکپارچه کرد و قادر به پردازش حداقل ۱۰ میلیون چندضلعی در

ثانیه بود. این پیشرفت، جهشی قابل توجه در عملکرد بازی‌های سه‌بعدی رایانه‌های شخصی ایجاد کرد. DirectX 8.0 در نوامبر ۲۰۰۰، برنامه‌ریزی‌پذیری را از طریق شیدرهای رأس و پیکسل معرفی کرد که برنامه‌نویسان را از ردیابی دستی وضعیت سخت‌افزار رها ساخت و یک تغییر بنیادی در برنامه‌نویسی گرافیک ایجاد نمود. Direct3D 9 در دسامبر ۲۰۰۲، زبان شیدینگ سطح بالا را بهبود بخشید و از فرمت‌های بافت ممیز شناور پشتیبانی کرد.

پیشرفت‌های معماری NVIDIA

انویدیا (NVIDIA) در سال ۱۹۹۳ توسط جن-سون هوانگ، کریس مالاچوفسکی، و کرتیس پریِم در ایالت کالیفرنیای آمریکا تأسیس شد. هدف اولیه این شرکت طراحی پردازنده‌های گرافیکی (GPU) برای کامپیوترهای شخصی و بازی‌های ویدئویی بود. در آن زمان، بازار پردازش گرافیکی در حال رشد بود و نیاز به سخت‌افزارهایی با قدرت پردازشی بالا به شدت احساس می‌شد. انویدیا با تمرکز بر نوآوری و تحقیق و توسعه، خیلی زود جایگاه خود را در صنعت تثبیت کرد و با عرضه‌ی اولین پردازنده گرافیکی خود به نام NV1 در سال ۱۹۹۵ وارد بازار شد، هرچند این محصول با استقبال چندانی مواجه نشد.

نقطه عطف فعالیت‌های انویدیا در سال ۱۹۹۹ رقم خورد، زمانی که این شرکت کارت گرافیک GeForce 256 را معرفی کرد؛ محصولی که به عنوان «اولین پردازنده گرافیکی جهان» شناخته می‌شود. این کارت گرافیک قادر بود پردازش‌های گرافیکی پیچیده را به‌صورت مستقل از CPU انجام دهد و باعث افزایش چشمگیر کیفیت بازی‌ها و کاربردهای تصویری شد. از آن زمان تاکنون، انویدیا با عرضه‌ی سری‌های متوالی از کارت‌های GeForce مانند سری‌های ۶۰۰۰، ۹۰۰۰، GTX، RTX 2000، ۳۰۰۰ و ۴۰۰۰، همواره پیشگام در توسعه فناوری‌های گرافیکی بوده است. هر نسل از این کارت‌ها، با بهبودهایی در معماری، حافظه، توان پردازشی و قابلیت‌های ردیابی پرتو (Ray Tracing) همراه بوده است.

انویدیا تنها به صنعت بازی محدود نماند و با معرفی معماری‌های CUDA و Tensor Core، جای خود را در زمینه‌های دیگر از جمله هوش مصنوعی، یادگیری عمیق، شبیه‌سازی‌های علمی، رندرینگ سه‌بعدی، و پردازش‌های موازی باز کرد. امروزه پردازنده‌های گرافیکی این شرکت نه تنها در رایانه‌های شخصی، بلکه در ابرکامپیوترها، دیتاسنترها، خودروهای خودران، و سیستم‌های هوش مصنوعی نیز کاربرد دارند. انویدیا همچنین با خرید شرکت‌هایی مانند Mellanox و ARM (در صورت نهایی شدن) قصد دارد حضور خود را در حوزه‌های پردازشی گسترده‌تری گسترش دهد. Fermi (۲۰۱۰)

این معماری آغازگر عصر محاسبات عمومی روی GPU بود و به عنوان "اولین معماری کامل محاسباتی GPU" شناخته می‌شود. Fermi دارای ۳ میلیارد ترانزیستور بود و هر Streaming Multiprocessor (SM) آن شامل ۳۲ هسته CUDA بود. این معماری از حافظه GDDR5 تا ۶ گیگابایت پشتیبانی می‌کرد و

یک کش L2 یکپارچه را معرفی نمود. Fermi همچنین زمان‌بند دوگانه Warp را برای اجرای همزمان دو گروه از رشته‌ها (warps) ارائه داد.

Kepler (۲۰۱۲)

Kepler بر بهره‌وری انرژی و برنامه‌ریزی‌پذیری تمرکز داشت. این معماری، SMX (نسل بعدی Streaming Multiprocessor) را معرفی کرد که تعداد هسته‌های CUDA را به ۱۹۲ هسته در هر SMX افزایش داد. از ویژگی‌های برجسته آن می‌توان به Dynamic Parallelism (قابلیت کرنل‌ها برای راه‌اندازی کرنل‌های دیگر) و Hyper-Q (افزایش صفوف کاری سخت‌افزاری از ۱ به ۳۲ برای بهره‌برداری بالاتر از GPU) اشاره کرد.

Maxwell (۲۰۱۴)

Maxwell با تمرکز بر بهره‌وری انرژی، طراحی Streaming Multiprocessor (SM) را بهبود بخشید و آن را SMM نامید. این معماری، کش L2 را به طور قابل توجهی (از ۲۵۶ کیلوبایت در Kepler به ۲ مگابایت) افزایش داد تا نیاز به پهنای باند حافظه را کاهش دهد. همچنین فناوری‌های رندرینگ جدیدی مانند Dynamic Super Resolution و VXGI (Voxel Global Illumination) را معرفی کرد.

Pascal (۲۰۱۶)

Pascal با استفاده از فناوری ساخت ۱۶ نانومتری FinFET، بهره‌وری انرژی بی‌سابقه‌ای را ارائه داد. این معماری، NVLink را برای مقیاس‌پذیری چند GPU (۵ برابر سریع‌تر از PCIe Gen5) و CoWoS با HBM2 را برای بارهای کاری داده‌های بزرگ (۷۲۰ گیگابایت بر ثانیه پهنای باند) یکپارچه کرد. Pascal همچنین دستورالعمل‌های ممیز شناور ۱۶ بیتی (FP16) را برای الگوریتم‌های هوش مصنوعی معرفی کرد که ۲۱ ترافلاپس برای آموزش و ۴۷ TOPS برای استنتاج ارائه می‌داد.

Volta (۲۰۱۷)

Volta یک معماری مهم بود که Tensor Cores را معرفی کرد؛ واحدهای تخصصی برای شتاب‌دهی عملیات ماتریسی که برای یادگیری عمیق حیاتی هستند. این هسته‌ها آموزش با دقت ترکیبی (ورودی‌های FP16 با خروجی‌های FP16/FP32) را ممکن ساختند و توان عملیاتی محاسباتی را به طور چشمگیری افزایش دادند (تا ۱۲ برابر افزایش ترافلاپس نسبت به Pascal). همچنین از NVLink 2.0 (۳۰۰ گیگابایت بر ثانیه پهنای باند GPU-GPU) بهره می‌برد.

Turning (۲۰۱۸)

Turing Ray Tracing Cores (هسته‌های RT) را برای شتاب‌دهی ردیابی پرتو در زمان واقعی معرفی کرد و برای اولین بار Tensor Cores را به GPUهای مصرف‌کننده آورد. همچنین فناوری DLSS یا

(Deep Learning Super Sampling) را معرفی کرد که از هوش مصنوعی برای مقیاس‌بندی و بهبود کیفیت تصویر استفاده می‌کند.

فناوری DLSS

DLSS که مخفف Deep Learning Super Sampling است، یکی از پیشرفته‌ترین تکنیک‌های رندرینگ مبتنی بر هوش مصنوعی محسوب می‌شود که توسط شرکت NVIDIA توسعه یافته است. این فناوری با بهره‌گیری از شبکه‌های عصبی عمیق و یادگیری ماشین، امکان اجرای بازی‌های ویدئویی و برنامه‌های گرافیکی سنگین را با نرخ فریم بالاتر و کیفیت تصویری نزدیک به رزولوشن اصلی فراهم می‌سازد. ایده اصلی در پشت DLSS آن است که تصویر نهایی با کیفیت بالا را از یک ورودی با رزولوشن پایین‌تر بازسازی کند؛ بدین ترتیب کارت گرافیک بخش سنگین رندرینگ را در رزولوشن پایین انجام می‌دهد و سپس با کمک الگوریتم‌های یادگیری عمیق، خروجی نهایی را با کیفیت بسیار نزدیک یا حتی گاه بهتر از رندر بومی ارائه می‌کند.

DLSS به‌صورت خاص بر روی کارت‌های گرافیک سری RTX انویدیا قابل اجرا است، چرا که برای پردازش شبکه عصبی نیاز به هسته‌های تنسوری (Tensor Cores) دارد که در این سری از کارت‌ها تعبیه شده‌اند. این هسته‌ها طراحی شده‌اند تا عملیات ماتریسی سنگین موردنیاز در یادگیری عمیق را با کارایی بالا انجام دهند. در نسخه‌های اولیه مانند DLSS 1.0، الگوریتم بر اساس یادگیری آفلاین و پردازش بر پایه تصویر ثابت عمل می‌کرد و نتایج آن وابسته به آموزش مدل برای هر بازی خاص بود. به همین دلیل، اگر مدل برای آن بازی خاص آموزش ندیده بود، کیفیت تصویر ممکن بود کاهش یابد یا جزئیات مهم از بین برود.

اما با عرضه نسخه‌های بعدی، به‌ویژه DLSS 2.0، معماری داخلی این فناوری به طرز چشم‌گیری تغییر کرد. در این نسخه، استفاده از شبکه عصبی بازآموزی‌شده و عمومی باعث شد دیگر نیازی به آموزش اختصاصی برای هر بازی نباشد. DLSS 2.0 با استفاده از اطلاعات فریم قبلی، بردارهای حرکتی و عمق صحنه، تصویری بسیار دقیق‌تر، شفاف‌تر و روان‌تر تولید می‌کند. نتیجه این پیشرفت، ارائه کیفیتی بسیار نزدیک به رندر اصلی در رزولوشن‌های بالا (مثلاً K۴) با بار پردازشی کمتر و نرخ فریم بیشتر است، که به‌ویژه برای نمایشگرهای گیمینگ و دستگاه‌های دارای توان محدود، یک پیشرفت بزرگ محسوب می‌شود.

نسخه سوم این فناوری یعنی DLSS 3.0 که هم‌زمان با معرفی معماری Ada Lovelace عرضه شد، یک گام فراتر رفت و مفهومی تحت عنوان Frame Generation را معرفی کرد. در این نسخه، فناوری DLSS قادر است فریم‌های کامل جدید را بین دو فریم اصلی تولید کند؛ به بیان دیگر، حتی در صورت عدم انجام کامل رندر سنتی توسط GPU، هوش مصنوعی می‌تواند یک فریم مجازی خلق کند که برای چشم انسان تقریباً از فریم واقعی قابل تمایز نیست. این ویژگی باعث افزایش شدید نرخ فریم در بازی‌ها می‌شود،

به‌ویژه در عناوینی که بار پردازشی بالایی دارند. البته این فناوری هنوز چالش‌هایی نیز دارد، از جمله تأخیر ورودی بالاتر یا اثرات بصری مصنوعی در برخی موقعیت‌ها، ولی پیشرفت آن در هر نسخه قابل‌توجه بوده است.

فناوری DLSS علاوه بر ارتقای عملکرد، تأثیرات مستقیمی بر مصرف انرژی و خنک‌سازی سیستم نیز دارد. چون بار پردازشی به‌صورت مؤثری کاهش پیدا می‌کند، مصرف توان الکتریکی پایین‌تر رفته و گرمای کمتری تولید می‌شود. این ویژگی باعث شده است که DLSS در لپ‌تاپ‌های گیمینگ نیز اهمیت ویژه‌ای پیدا کند، زیرا می‌توان عملکردی در سطح سیستم‌های دسکتاپ را با محدودیت‌های سخت‌افزاری کمتر تجربه کرد. همچنین، استفاده از DLSS به توسعه‌دهندگان بازی این امکان را می‌دهد که جلوه‌های بصری سنگین‌تری مانند Ray Tracing را در بازی‌ها پیاده‌سازی کنند، بدون اینکه عملکرد کلی بازی قربانی شود.

در نهایت، DLSS به‌عنوان ترکیبی از هوش مصنوعی و گرافیک کامپیوتری، نمونه‌ای درخشان از کاربرد یادگیری عمیق در حل مشکلات واقعی صنعت گیمینگ است. این فناوری نشان می‌دهد که آینده رندرینگ گرافیکی نه صرفاً به افزایش قدرت خام پردازنده‌های گرافیکی، بلکه به روش‌های هوشمندانه و محاسباتی وابسته است. با رقابت در این حوزه از سوی سایر شرکت‌ها (مانند AMD با FSR و Intel با XeSS)، می‌توان انتظار داشت که فناوری‌های مشابهی با رویکردهای بازتر و گسترده‌تر در آینده نقش پررنگ‌تری در توسعه و بهینه‌سازی تجربه بصری کاربران ایفا کنند.

Ampere (۲۰۲۰)

Ampere دارای Tensor Cores نسل سوم (۲ تا ۴ برابر توان عملیاتی، پشتیبانی از TF32 و BF32، و پراکندگی ساختاریافته ریزدانه) و Ray Tracing Cores نسل دوم (۲ برابر عملکرد نسل اول) بود. این معماری امکان اجرای همزمان بارهای کاری هسته‌های RT و CUDA را فراهم کرد. توان عملیاتی A100 را از ۹۰۰ گیگابایت بر ثانیه در V100 به ۱۵۵۵ گیگابایت بر ثانیه افزایش داد.

Ada Lovelace (۲۰۲۲)

Ada Lovelace هسته‌های RT نسل سوم (۲ برابر عملکرد ردیابی پرتو نسبت به نسل قبل) و Tensor Cores نسل چهارم (تا ۴ برابر توان استنتاج بالاتر با دقت FP8 و پراکندگی ساختاریافته) را معرفی کرد. این معماری همچنین DLSS 3 را با Frame Generation (تولید فریم‌های کاملاً جدید با هوش مصنوعی) به ارمغان آورد.

Hopper (۲۰۲۲) و Blackwell (۲۰۲۵/۲۰۲۴)

این معماری‌ها که عمدتاً برای مراکز داده طراحی شده‌اند، هوش مصنوعی و HPC را به پیش می‌برند. Hopper موتور Transformer را معرفی کرد (تا ۹ برابر آموزش هوش مصنوعی سریع‌تر، ۳۰ برابر استنتاج سریع‌تر برای مدل‌های زبان بزرگ (LLMs))، Confidential Computing و NVLink

Switch را نیز شامل می‌شود. Blackwell (سری RTX 50 برای مصرف‌کنندگان) برای رندرینگ عصبی ساخته شده است و دارای ۲۰۸ میلیارد ترانزیستور (۲.۵ برابر Hopper)، یک کلاس جدید از ابرتراشه‌های هوش مصنوعی با دو دای متصل به هم از طریق NV-HBI با سرعت ۱۰ ترابایت بر ثانیه، موتور Transformer نسل دوم (پشتیبانی از FP4، ۲ برابر توان عملیاتی Tensor Core برای FP4)، Confidential Computing، NVLink نسل پنجم (۱.۸ ترابایت بر ثانیه در هر GPU) و موتور Decompression است. هسته‌های RT در Blackwell۸ برابر سریع‌تر از Turing در فرکانس یکسان هستند و هسته‌های CUDA آن برای شیدینگ عصبی بازطراحی شده‌اند.

پیشرفت‌های معماری AMD

شرکت ای‌ام‌دی (AMD یا Advanced Micro Devices) در سال ۱۹۶۹ توسط جری ساندرز و چند مهندس دیگر که از شرکت Fairchild Semiconductor جدا شده بودند، در ایالت کالیفرنیای آمریکا تأسیس شد. در ابتدا تمرکز این شرکت بر طراحی و تولید تراشه‌ها و نیمه‌رساناها برای کاربردهای مختلف بود. ای‌ام‌دی در دهه‌های ابتدایی فعالیتش بیشتر به عنوان تولیدکننده‌ی پردازنده‌های مرکزی (CPU) شناخته می‌شد و رقیب مستقیم اینتل در بازار پردازنده‌ها بود. با این حال، این شرکت با خرید شرکت ATI Technologies در سال ۲۰۰۶، وارد بازار پردازنده‌های گرافیکی (GPU) شد و از آن زمان تاکنون به عنوان یکی از دو قطب اصلی صنعت کارت گرافیک در کنار انویدیا شناخته می‌شود.

پس از خرید ATI، شرکت AMD تولید کارت‌های گرافیکی را با برند Radeon آغاز کرد. سری‌های اولیه مانند Radeon HD 4000 و ۵۰۰۰ با استقبال خوبی مواجه شدند و توانستند سهم مناسبی از بازار گیمرها را به خود اختصاص دهند. AMD در این کارت‌ها تمرکز ویژه‌ای بر نسبت قیمت به کارایی داشت و معمولاً در بازه‌های قیمتی میان‌رده و اقتصادی، رقابتی‌تر از انویدیا عمل می‌کرد. در سال‌های بعد، سری‌های R7، R9، RX 400، ۵۰۰، و سری‌های قدرتمند RX Vega و RX 5000 و ۶۰۰۰ با معماری RDNA وارد بازار شدند. در این مسیر، AMD تلاش کرد تا با ارائه معماری‌های جدید، مصرف انرژی را بهینه کند و قابلیت‌هایی مانند Ray Tracing و Smart Access Memory را به رقابت با فناوری‌های انویدیا بفرستد.

در سال‌های اخیر، AMD با معماری RDNA 3 و سری‌های Radeon RX 7000 توانسته جایگاه خود را در بازار کارت گرافیک‌های رده بالا نیز تثبیت کند. از سوی دیگر، با تمرکز بر طراحی‌های مبتنی بر چیپلت (chiplet-based design)، AMD نوآوری‌هایی در زمینه تولید پردازنده‌های گرافیکی و ترکیبی (APU) به کار گرفته است. این شرکت همچنین به‌طور گسترده در بازار کنسول‌های بازی حضور دارد و پردازنده‌های گرافیکی و مرکزی کنسول‌های نسل جدید مانند پلی‌استیشن ۵ و ایکس‌باکس سری X را تأمین می‌کند. حضور همزمان AMD در حوزه‌های CPU و GPU باعث شده که بتواند محصولات ترکیبی و هماهنگی را به بازار عرضه کند که در کاربردهایی مانند بازی، تولید محتوا و حتی محاسبات علمی کاربرد زیادی دارند.

TeraScale (۲۰۰۵-۲۰۱۳)

معماری TeraScale اولین معماری پردازش گرافیکی مدرن AMD پس از دوران ATI محسوب می‌شود که با معرفی سری Radeon HD 2000 در سال ۲۰۰۷ آغاز شد. این معماری بر اساس طراحی VLIW (Very Long Instruction Word) بنا شده بود که چندین دستور را به صورت موازی در یک واحد پردازشی اجرا می‌کرد. در این معماری، تمرکز اصلی بر افزایش توان عملیاتی از طریق موازی‌سازی و پردازش چند رشته‌ای بود، اما پیچیدگی در زمان‌بندی و توزیع دستورها باعث شد در برخی سناریوها کارایی پایین‌تری نسبت به رقبا داشته باشد.

معماری TeraScale در طول زمان بهبودهایی به خود دید و به نسخه‌های بعدی مانند TeraScale 2 (سری Radeon HD 5000) و TeraScale 3 (سری Radeon HD 6000) ارتقا یافت. در نسخه‌های جدیدتر، AMD تلاش کرد با بهینه‌سازی طراحی VLIW، ضعف‌های مربوط به تخصیص دستورها و استفاده ناقص از منابع پردازشی را کاهش دهد. به‌ویژه سری HD 5000 که اولین کارت‌های گرافیکی با پشتیبانی از DirectX 11 را ارائه دادند، با استقبال بسیار خوبی در بازار مواجه شدند و آغازگر موفقیت AMD در دنیای کارت‌های DirectX 11 شدند.

با وجود پیشرفت‌ها، محدودیت‌های ساختاری VLIW همچنان به عنوان گلوگاه عملکرد باقی ماند. در کارهای عمومی مانند محاسبات علمی (GPGPU) و موتورهای گرافیکی پیچیده، این معماری نتوانست از تمام توان بالقوه سخت‌افزار استفاده کند. این چالش‌ها AMD را مجبور به تغییر اساسی در فلسفه طراحی کرد که در نهایت منجر به کنار گذاشتن TeraScale و خلق معماری GCN در سال ۲۰۱۱ شد

GCN (۲۰۱۲-۲۰۲۲)

معماری GCN در سال ۲۰۱۱ با معرفی کارت‌های Radeon HD 7000 عرضه شد و نقطه عطفی در طراحی GPUهای AMD محسوب می‌شود. برخلاف معماری VLIW، در GCN ساختار به سمت SIMD (Single Instruction, Multiple Data) و بهره‌وری بالاتر در پردازش‌های عمومی و گرافیکی سوق داده شد. هر واحد پردازشی در GCN به گونه‌ای طراحی شده بود که بتواند به شکل مستقل، داده‌ها را پردازش کند و توانایی اجرای موازی بیشتری نسبت به TeraScale داشت. همچنین این معماری برای استفاده در محیط‌های غیرگرافیکی مانند OpenCL و HSA (Heterogeneous System Architecture) نیز بهینه‌سازی شده بود.

GCN در چندین نسل و بهبود متوالی عرضه شد: GCN 1.0 (HD 7000)، GCN 2.0 (Radeon R9 200)، GCN 3.0 (R9 300/Fury)، و GCN 4.0 (Polaris)، و نهایتاً نسخه ۵.۰ (Vega) که آخرین نسخه این معماری بود. هر نسل با بهبودهایی در کش‌ها، بهینه‌سازی مصرف انرژی، افزایش پهنای باند حافظه و افزایش تعداد هسته‌های پردازشی همراه بود. کارت‌های مشهور مبتنی بر GCN مانند

Radeon R9 290X، RX 480 و RX Vega 64 نشان دادند که AMD می‌تواند در رده‌های مختلف، از میان‌رده تا پرچم‌دار، با انویدیا رقابت کند.

اگرچه GCN از نظر توان محاسباتی قدرتمند بود، اما مصرف انرژی نسبتاً بالا و طراحی ماژولار پیچیده باعث شد که در بازی‌های سبک جدید و بارهای گرافیکی سنگین، در برابر انویدیا کمی عقب بماند. AMD نیز به تدریج به این نتیجه رسید که برای رسیدن به کارایی بهتر در بازی‌ها و کاهش مصرف انرژی باید معماری جدیدی طراحی کند. این تصمیم در نهایت به خلق RDNA منجر شد؛ معماری‌ای که به صورت تخصصی برای بازی و بهره‌وری انرژی ساخته شده بود.

RDNA (Radeon DNA) (۲۰۱۹–حال)

AMD در سال ۲۰۱۹ معماری RDNA را به عنوان جانشین GCN معرفی کرد. این معماری با کارت‌های سری Radeon RX 5000 و معماری RDNA 1.0 (مانند RX 5700 XT) رونمایی شد. RDNA بر خلاف GCN که برای پردازش‌های عمومی نیز طراحی شده بود، تمرکز ویژه‌ای بر بازی‌های رایانه‌ای، بهره‌وری انرژی، و تأخیر پایین‌تر در اجرای دستورات داشت. از مهم‌ترین تغییرات RDNA می‌توان به طراحی مجدد واحدهای پردازشی (CU)، معماری جدید کش، و بهینه‌سازی مسیر داده اشاره کرد. نتیجه‌ی این تغییرات، افزایش ۵۰ درصدی عملکرد به ازای هر وات نسبت به نسل قبلی بود.

با معرفی RDNA 2 در سال ۲۰۲۰ و کارت‌هایی نظیر Radeon RX 6800 و RX 6900 XT، AMD پشتیبانی از فناوری‌هایی مانند Ray Tracing سخت‌افزاری، Infinity Cache، و Smart Access Memory را اضافه کرد. این نسل از نظر عملکرد بسیار نزدیک به کارت‌های سری RTX 3000 انویدیا بود و در بسیاری از موارد توانست در رقابت مستقیم موفق عمل کند. همچنین RDNA 2 در کنسول‌های نسل نهم مانند PS5 و Xbox Series X نیز استفاده شد، که به گسترش حضور AMD در بازار بازی‌های کنسولی کمک شایانی کرد.

در سال ۲۰۲۲ و ۲۰۲۳، AMD نسل سوم این معماری را با نام RDNA 3 معرفی کرد که کارت‌هایی نظیر RX 7900 XTX را شامل می‌شود. از مهم‌ترین نوآوری‌های این نسل می‌توان به استفاده از معماری چیپلت (chiplet-based design)، افزایش ظرفیت کش، و بهبود قابل توجه در ray tracing اشاره کرد. RDNA 3 همچنان بر بهره‌وری انرژی بالا و عملکرد گیمینگ تمرکز دارد و موقعیت AMD را در بازار کارت‌های گرافیکی رده‌بالا تثبیت کرده است. این روند نشان می‌دهد که AMD قصد دارد معماری RDNA را برای چندین نسل آینده نیز توسعه دهد.

RDNA 4.0 (۲۰۲۵)

معماری RDNA 4 که نسل بعدی معماری گرافیکی بازی محور AMD پس از RDNA 3 محسوب می‌شود، به عنوان نسخه‌ای ارتقاءیافته برای بهبود عملکرد در بازی‌های رایانه‌ای و بهره‌وری انرژی طراحی شده است. RDNA 4 با تمرکز بر ارتقای معماری چیپلت (chiplet-based) که در RDNA 3 معرفی شده

بود، سعی دارد کارایی در هر وات مصرف انرژی را بیش از پیش افزایش دهد. طبق اطلاعات فنی و گزارش‌های منتشرشده، RDNA 4 قرار است بهبودهایی در واحدهای محاسباتی (CU)، کش‌ها، واحدهای شتاب‌دهنده Ray Tracing، و معماری مسیر داده (data path) ارائه دهد.

یکی از ویژگی‌های احتمالی RDNA 4، استفاده از نود ساخت جدید (مانند ۴ نانومتری یا ترکیب ۴ و ۳ نانومتری برای چیپلت‌ها) است که منجر به تراکم ترانزیستور بالاتر و مصرف انرژی پایین‌تر می‌شود. همچنین انتظار می‌رود AMD در این نسل، حافظه‌های سریع‌تر GDDR7 را جایگزین GDDR6 یا GDDR6X کند و Infinity Cache را از نظر ظرفیت و پهنای باند بهبود دهد. با توجه به رقابت شدید با انویدیا در بازار گرافیک‌های بالارده و میان‌رده، RDNA 4 احتمالاً روی ارائه تجربه بازی روان در وضوح‌های K۴ و حتی K۸ تمرکز خواهد داشت.

واحدهای محاسباتی بازنگری‌شده با معماری دوگانه SIMD32 و عملیات ماتریسی بهبودیافته (۲ برابر نرخ ماتریس ۱۶ بیتی، ۴ برابر نرخ ماتریس ۴/۸ بیتی، پراکندگی ساختاریافته ۴:۲، انواع داده ممیز شناور ۸ بیتی جدید) را معرفی می‌کند. واحدهای ردیابی پرتو نسل سوم با نرخ‌های برش دو برابر و فشرده‌سازی BVH بهبودیافته همراه هستند. این معماری ویژگی‌های هوش مصنوعی را برای مقیاس‌بندی و رندرینگ، از جمله نمونه‌برداری عصبی و حذف نویز، یکپارچه می‌کند و ۳.۵ برابر افزایش در محاسبات FP16 ML در هر CU نسبت به RDNA 2 را به ارمغان می‌آورد. FSR 4 از یک الگوریتم مقیاس‌بندی شتاب‌یافته با هوش مصنوعی استفاده می‌کند.

CDNA (۲۰۲۰-حال)

معماری CDNA که مخفف Compute DNA است، در سال ۲۰۲۰ توسط AMD معرفی شد و هدف آن ارائه یک معماری بهینه‌شده برای محاسبات با کارایی بالا (HPC)، مراکز داده، یادگیری ماشین و هوش مصنوعی بود. برخلاف RDNA که تمرکز آن بر گرافیک و بازی است، CDNA به‌طور کامل فاقد اجزای گرافیکی سنتی مانند rasterizer یا units گرافیکی است و فقط برای محاسبه بهینه شده است. اولین نسخه از این معماری، CDNA 1، با کارت‌های Instinct MI100 روانه بازار شد.

معماری CDNA از طراحی بسیار پیشرفته‌ای بهره می‌برد که بر پایه‌ی هسته‌های ماتریسی (Matrix Cores) و پشتیبانی از FP16، bfloat16، INT8 و سایر فرمت‌های داده‌ای تخصصی برای یادگیری ماشین بنا شده است. همچنین این معماری با استفاده از Infinity Fabric امکان ارتباط سریع بین چند GPU را فراهم می‌سازد، که در کارهایی مثل شبیه‌سازی آب‌وهوا، مدل‌سازی ژنتیک، دینامیک سیالات محاسباتی (CFD)، و شبکه‌های عصبی عمیق بسیار کاربردی است. AMD در این معماری از HBM2e memory با پهنای باند بسیار بالا استفاده کرده که برای پردازش‌های حافظه‌محور ضروری است.

نسل دوم این معماری، یعنی CDNA 2، در سال ۲۰۲۱ با کارت‌های Instinct MI200 عرضه شد. در این نسل، برای نخستین بار طراحی چیپلت نیز وارد دنیای پردازنده‌های گرافیکی محاسباتی شد. کارت‌های

MI250X مبتنی بر این معماری عملکردی فوق‌العاده در محاسبات شناور و یادگیری ماشین از خود نشان دادند. AMD در نسل بعدی یعنی CDNA 3 که در سال ۲۰۲۴ معرفی شده، تمرکز بیشتری بر هوش مصنوعی به‌ویژه در مدل‌های زبانی بزرگ (LLMها) داشته است. این معماری برای رقابت با NVIDIA H100 و H200 و شتاب‌دهنده‌های Google TPU طراحی شده و در دیتاسنترها، ابرکامپیوترها، و زیرساخت‌های ابری کاربرد فراوانی دارد.

فناوری FSR

فناوری FSR که مخفف FidelityFX Super Resolution است، یکی از پیشرفته‌ترین تکنیک‌های ارتقاء مقیاس (upscaling) در دنیای گرافیک رایانه‌ای به شمار می‌رود که توسط شرکت AMD توسعه یافته است. این فناوری با هدف بهبود نرخ فریم و حفظ کیفیت تصویر در بازی‌های ویدیویی ارائه شده و به عنوان پاسخ AMD به فناوری DLSS شرکت NVIDIA شناخته می‌شود. بر خلاف DLSS که به سخت‌افزار خاص (هسته‌های تنسوری) متکی است، FSR بر پایه الگوریتم‌های مبتنی بر پردازش تصویری اجرا می‌شود و با طیف وسیعی از سخت‌افزارها – حتی کارت‌های گرافیک غیر AMD – سازگار است. همین ویژگی، FSR را به گزینه‌ای محبوب برای توسعه‌دهندگان بازی و گیمرهایی با سخت‌افزار متنوع تبدیل کرده است.

نخستین نسخه این فناوری، FSR 1.0، در سال ۲۰۲۱ معرفی شد و از یک الگوریتم spatial upscaling بهره می‌برد. در این روش، اطلاعات مربوط به هر فریم به صورت مجزا و بدون در نظر گرفتن فریم‌های قبلی پردازش می‌شود. این نوع مقیاس‌گذاری، اگرچه ساده‌تر و کم‌هزینه‌تر از نظر محاسباتی است، اما در بازسازی جزئیات ریز و کاهش مصنوعات تصویری محدودیت‌هایی دارد. FSR 1.0 به دلیل ساختار باز و سادگی پیاده‌سازی، به سرعت در بین بازی‌سازها مورد استقبال قرار گرفت و در مدت کوتاهی در تعداد زیادی از بازی‌ها به‌کار گرفته شد.

با ورود به نسل دوم، یعنی FSR 2.0، AMD تغییرات بنیادینی در معماری این فناوری اعمال کرد. برخلاف نسخه پیشین، FSR 2.0 یک الگوریتم temporal upscaling است که از اطلاعات فریم‌های قبلی، بردارهای حرکتی (motion vectors) و داده‌های عمق (depth buffer) برای بازسازی فریم با وضوح بالا استفاده می‌کند. این تحول بزرگ باعث شد FSR 2.0 قادر باشد کیفیت تصویری بسیار نزدیک به رندر بومی را در خروجی ارائه دهد. استفاده از روش‌های پیشرفته‌تر در FSR 2.0 منجر به کاهش محسوس مصنوعات بصری مانند لرزش لبه‌ها، شبح‌زدگی و پارگی تصویر شده و تجربه بصری روان‌تری فراهم می‌کند.

یکی از مزیت‌های کلیدی FSR نسبت به برخی فناوری‌های مشابه، عدم نیاز به سخت‌افزار اختصاصی و خاص است. این موضوع به توسعه‌دهندگان امکان می‌دهد که از FSR در پلتفرم‌های گوناگون، شامل کنسول‌های بازی، کارت‌های گرافیک قدیمی‌تر، و حتی رقبا مانند کارت‌های NVIDIA استفاده کنند. این

استقلال سخت‌افزاری، همراه با ساختار متن‌باز و مستندات جامع، باعث شده است که FSR تبدیل به ابزاری گسترده و مقیاس‌پذیر برای بهینه‌سازی عملکرد در بازی‌ها شود.

در ادامه، AMD نسخه FSR 3.0 را معرفی کرد که در آن فناوری جدیدی به نام Frame Generation به کار گرفته شده است. این فناوری مشابه چیزی است که پیش‌تر در DLSS 3.0 دیده شد، و به GPU اجازه می‌دهد تا با استفاده از الگوریتم‌های پیش‌بینی، فریم‌های میانی را بین فریم‌های اصلی تولید کند. این روش می‌تواند بدون افزایش قابل‌توجه بار رندرینگ، نرخ فریم را به شکل چشمگیری بالا ببرد. البته این تکنیک همچنان در حال بلوغ است و در برخی موارد می‌تواند منجر به کاهش دقت حرکات سریع یا ایجاد مصنوعات تصویری شود، ولی پیشرفت‌های مداوم در آن قابل توجه‌اند.

از منظر عملکرد، FSR راهی ساده و کارآمد برای دستیابی به تعادل بین کیفیت بصری و نرخ فریم در بازی‌های مدرن فراهم می‌کند. گیمرها با استفاده از این فناوری می‌توانند رزولوشن پایه را کاهش داده و با کمک ارتقاء هوشمند تصویر، کیفیت نهایی را حفظ کنند، بدون اینکه فشار سنگینی به GPU وارد شود. این امر به‌ویژه در سیستم‌های میان‌رده یا لپ‌تاپ‌ها حائز اهمیت است، جایی که منابع پردازشی محدودترند و مدیریت انرژی و حرارت اهمیت بیشتری دارد.

ورود Intel به بازار GPU

ورود شرکت Intel به بازار پردازنده‌های گرافیکی (GPU) یکی از مهم‌ترین تحولات اخیر در صنعت سخت‌افزار رایانه‌ای به شمار می‌رود. شرکتی که دهه‌ها به‌عنوان بازیگر اصلی در طراحی و تولید پردازنده‌های مرکزی (CPU) شناخته می‌شد، با هدف تنوع‌بخشی به سبد محصولات خود، افزایش رقابت‌پذیری، و پاسخ به نیاز روزافزون بازار برای پردازش‌های گرافیکی و محاسباتی، تصمیم گرفت به طور جدی وارد عرصه تولید کارت‌های گرافیکی مجزا شود. این حرکت استراتژیک نشان‌دهنده درک عمیق اینتل از تغییر مسیر فناوری در دهه‌های آینده است؛ جایی که هوش مصنوعی، بازی‌های رایانه‌ای، رندرینگ سه‌بعدی و شبیه‌سازی‌های علمی بیش از پیش به قدرت پردازش گرافیکی متکی هستند.

اولین گام رسمی و جدی اینتل در این حوزه با معرفی معماری گرافیکی جدید به نام Xe برداشته شد. این معماری در واقع پایه و اساس تمامی تلاش‌های اینتل در حوزه گرافیک بود و به گونه‌ای طراحی شده است که در مقیاس‌های مختلف – از گرافیک مجتمع در لپ‌تاپ‌های سبک گرفته تا کارت‌های گرافیکی قدرتمند برای مراکز داده – قابل استفاده باشد. خانواده Xe شامل زیرمعماری‌هایی چون Xe-LP (برای محصولات سبک)، Xe-HPG (برای بازی و گرافیک با کارایی بالا)، Xe-HPC (برای محاسبات سنگین علمی)، و Xe-HP (برای حجم کاری دیتاسنتری) می‌شود. این طراحی مقیاس‌پذیر و منعطف به اینتل اجازه داده است تا از ابتدا، گرافیک را نه‌تنها به عنوان مکمل پردازنده‌هایش، بلکه به عنوان حوزه‌ای مستقل و دارای چشم‌انداز بلندمدت ببیند.

در سال ۲۰۲۲، اینتل نخستین سری از کارت‌های گرافیکی مجزای خود را با برند Intel Arc روانه بازار کرد. سری Arc با دو مدل ابتدایی به نام‌های Arc A750 و Arc A770 معرفی شد که هدف آن‌ها رقابت با کارت‌های میان‌رده‌ی بازار از برندهای NVIDIA و AMD بود. این محصولات از فناوری‌هایی چون Ray Tracing در سطح سخت‌افزاری پشتیبانی می‌کردند و همچنین از تکنولوژی XeSS بهره می‌بردند؛ فناوری ارتقاء مقیاس مبتنی بر هوش مصنوعی که به‌صورت مشابه با DLSS انویدیا طراحی شده است. XeSS می‌تواند کیفیت تصویری نزدیک به رزولوشن بومی را با استفاده از اطلاعات فریم‌های قبلی و شبکه عصبی بازسازی کند، و در نتیجه عملکرد را بدون فدا کردن کیفیت، بهبود بخشد.

یکی از چالش‌های اصلی اینتل در ورود به بازار GPU، رقابت با دو غول بزرگ و قدیمی این حوزه یعنی NVIDIA و AMD بود. این دو شرکت با سابقه طولانی در توسعه فناوری‌های گرافیکی و برخورداری از اکوسیستم نرم‌افزاری کامل، از مزایای رقابتی بزرگی برخوردارند. اینتل، برای جبران این فاصله، سرمایه‌گذاری‌های کلانی در حوزه درایورها، توسعه‌دهندگان بازی و ابزارهای نرم‌افزاری کرده است. در ابتدای عرضه، بسیاری از کاربران و رسانه‌ها از ناپایداری درایورها و ناسازگاری نرم‌افزاری محصولات Arc انتقاد داشتند، اما اینتل به‌تدریج با انتشار به‌روزرسانی‌های منظم و بهینه‌سازی‌های متعددی، بخشی از این نواقص را برطرف کرد و عملکرد محصولات خود را بهبود بخشید.

حرکت اینتل به سمت GPUها تنها به بازی و گرافیک محدود نمی‌شود. این شرکت با معرفی GPUهای Xe-HPC و پروژه‌هایی مانند Ponte Vecchio، به وضوح نشان داد که درصدد رقابت در بازارهای محاسبات علمی، یادگیری ماشین، و مراکز داده نیز هست. این نوع پردازش‌ها نیازمند توان عظیم محاسباتی، پهنای باند بالا و معماری‌هایی با راندمان انرژی بالا هستند. ورود به این حوزه‌ها نشان‌دهنده جاه‌طلبی اینتل برای تبدیل شدن به بازیگری جامع در پردازش موازی است، جایی که GPU نقش کلیدی ایفا می‌کند.

یکی دیگر از نکات قابل‌توجه در استراتژی اینتل، تمرکز بر ادغام GPU با سایر فناوری‌های موجود در زنجیره محصولاتش است. برای نمونه، بهره‌گیری از PCIe 5.0، حافظه‌های DDR5 و فناوری‌های جدید مانند CXL، نشان می‌دهد اینتل قصد دارد با ایجاد هماهنگی عمیق میان CPU و GPU، عملکرد بهینه‌تری را ارائه دهد. همچنین، حرکت به سمت معماری‌های چند تراشه‌ای (multi-chip) و تولید GPUهایی با طراحی ماژولار، به اینتل امکان می‌دهد که رقابت‌پذیری خود را در سال‌های آتی افزایش دهد.

جمع بندی:

رویکرد AMD در تقسیم معماری به RDNA (گرافیک/بازی) و CDNA (HPC/AI) در تضاد با رویکرد NVIDIA است که واحدهای تخصصی (Tensor، RT) را در یک معماری یکپارچه‌تر ادغام می‌کند. این موضوع نشان‌دهنده رویکردهای استراتژیک متفاوتی برای خدمت‌رسانی به بازارهای متنوع است. رویکرد دو معماری AMD ممکن است بهینه‌سازی بیشتری را برای دامنه‌های خاص ارائه دهد، در حالی که رویکرد

یکپارچه NVIDIA به دنبال قابلیت کاربرد گسترده‌تر و تجربه توسعه‌دهنده یکپارچه‌تر (اکوسیستم CUDA) است. این پویایی رقابتی، راه‌حل‌های متنوعی را ترویج می‌دهد و مرزهای ممکن را جابجا می‌کند.

فناوری‌هایی مانند DLSS NVIDIA و FSR AMD صرفاً نوآوری‌های سخت‌افزاری یا نرم‌افزاری نیستند، بلکه ترکیبی محکم از هر دو هستند. DLSS به Tensor Cores نیاز دارد و FSR 4 اکنون از مقیاس‌بندی شتاب‌یافته با هوش مصنوعی استفاده می‌کند. این موضوع نشان می‌دهد که پیشرفت‌های مدرن GPU به طور فزاینده‌ای جامع هستند و نیازمند توسعه هم‌افزا در طراحی سیلیکون، ریزمعماری و الگوریتم‌های نرم‌افزاری پیشرفته می‌باشند. این وابستگی متقابل به این معناست که قدرت سخت‌افزاری خام به تنهایی کافی نیست؛ یک اکوسیستم نرم‌افزاری قوی برای آزادسازی عملکرد و فعال‌سازی ویژگی‌های جدید به همان اندازه حیاتی است.

مقایسه GPU با ASIC و FPGA

FPGA، یا آرایه گیت قابل برنامه‌ریزی در میدان (Field-Programmable Gate Array)، به مثابه قلب تپنده بسیاری از سیستم‌های دیجیتال مدرن عمل می‌کند که نیاز به انعطاف‌پذیری، سرعت بالا و قابلیت پیکربندی مجدد دارند. این قطعه سیلیکونی منحصربه‌فرد، پل ارتباطی قدرتمندی میان دنیای نرم‌افزار و سخت‌افزار ایجاد کرده است. FPGA به طراحان این امکان را می‌دهد تا مدارهای منطقی را حتی پس از تولید تراشه، بر اساس نیازهای خاص خود تعریف و پیاده‌سازی کنند. این قابلیت، FPGA را در مقابل مدارهای مجتمع با کاربرد خاص (ASIC)، که عملکردشان در زمان ساخت به صورت ثابت و غیرقابل تغییر حک می‌شود، به طور چشمگیری متمایز می‌سازد.

برای درک عمیق‌تر قابلیت‌های بی‌نظیر FPGA، کاوش در معماری داخلی آن ضروری است. یک FPGA از سه جزء اصلی و چندین بلوک تخصصی تشکیل شده است که هر یک نقش حیاتی در عملکرد نهایی آن ایفا می‌کنند. بلوک‌های منطقی قابل پیکربندی (CLBs) را می‌توان به عنوان عناصر بنیادی سازنده FPGA در نظر گرفت. هر CLB، ترکیبی از جداول جستجو (Look-Up Tables - LUTs) و فلیپ‌فلاپ‌ها (Flip-Flops) را در خود جای داده است. LUTها در حقیقت حافظه‌های کوچکی هستند که می‌توانند هر تابع منطقی بولی با تعداد ورودی مشخص را پیاده‌سازی کنند. به عنوان مثال، یک LUT چهار ورودی قادر است هر تابع منطقی تعریف شده با چهار ورودی را بازسازی کند، که این قابلیت انعطاف‌پذیری فوق‌العاده‌ای را در پیاده‌سازی گیت‌های منطقی پیچیده فراهم می‌آورد. فلیپ‌فلاپ‌ها نیز برای ذخیره‌سازی وضعیت و پیاده‌سازی مدارهای ترتیبی (sequential circuits) نظیر شمارنده‌ها و شیفت رجیسترها به کار می‌روند. ارتباط و همکاری میان LUTها و فلیپ‌فلاپ‌ها درون یک CLB، امکان ساخت بلوک‌های منطقی با پیچیدگی‌های بالاتر را فراهم می‌سازد.

جزء مهم دیگر، شبکه اتصال‌دهنده قابل برنامه‌ریزی (Programmable Interconnect) است. این جزء نقش "سیم‌کشی" داخلی FPGA را بر عهده دارد و وظیفه ارتباط بین CLBها، منابع I/O و سایر بلوک‌های تخصصی را ایفا می‌کند. این شبکه شامل مجموعه‌ای از خطوط مسی و سوئیچ‌های قابل برنامه‌ریزی است. با فعال یا غیرفعال کردن هوشمندانه این سوئیچ‌ها، می‌توان مسیرهای ارتباطی دلخواه را بین بلوک‌های مختلف ایجاد کرد و به این ترتیب، مدار منطقی مورد نظر را به هم متصل نمود. این انعطاف‌پذیری بی‌نظیر در اتصال، یکی از دلایل اصلی قدرت و کاربرد وسیع FPGA به شمار می‌رود. در نهایت، بلوک‌های ورودی/خروجی (Input/Output Blocks - IOBs) در حاشیه تراشه قرار گرفته‌اند و به عنوان واسطی حیاتی بین پین‌های فیزیکی FPGA و منطق داخلی آن عمل می‌کنند. IOBها قابل پیکربندی هستند تا استانداردهای ولتاژ مختلفی را پشتیبانی کنند (مانند LVCMOS، LVDS، SSTL و غیره) و همچنین می‌توانند برای عملکردهای خاصی مانند مقاومت‌های pull-up/down، قابلیت تری-استیت (tri-state) و یا پیاده‌سازی تاخیرهای دقیق کالیبره شوند.

علاوه بر این سه جزء اساسی، FPGAهای مدرن شامل بلوک‌های سخت‌افزاری تخصصی دیگری نیز هستند که به طرز چشمگیری عملکرد و کارایی کلی سیستم را افزایش می‌دهند. این بلوک‌ها شامل بلوک‌های DSP (Digital Signal Processing) برای پیاده‌سازی توابع پردازش سیگنال نظیر ضرب‌کننده‌ها و واحدهای ضرب‌کننده-جمع‌کننده، بلوک‌های حافظه داخلی (Block RAMs) با ظرفیت بالا و سرعت دسترسی سریع برای ذخیره‌سازی داده‌ها، ساعت‌های مدیریت کلاک (Clock Management Tiles - CMTs) شامل ابزارهایی مانند PLL و DCM برای تولید و توزیع سیگنال‌های ساعت با دقت بالا، و در برخی موارد، هسته‌های پردازنده (Processor Cores) مانند ARM هستند که به صورت سخت‌افزاری و مجتمع در تراشه وجود دارند. این ادغام، به طراحان اجازه می‌دهد تا همزمان از انعطاف‌پذیری منطق قابل برنامه‌ریزی و قدرت پردازشی یک پردازنده نرم‌افزاری در یک تراشه واحد بهره‌مند شوند، که منجر به طراحی سیستم‌های روی تراشه (SoC) بسیار قدرتمند می‌شود.

فرایند طراحی و پیاده‌سازی یک سیستم بر روی FPGA، یک چرخه تکراری و چندمرحله‌ای است که شامل مراحل کلیدی می‌شود. در مرحله ورود طرح (Design Entry)، طراح رفتار مدار مورد نظر خود را توصیف می‌کند. رایج‌ترین و مؤثرترین روش برای این کار استفاده از زبان‌های توصیف سخت‌افزار (Hardware Description Languages - HDLs) نظیر VHDL و Verilog است. این زبان‌ها امکان توصیف موازی‌سازی، همزمانی و ساختار سخت‌افزار را فراهم می‌کنند. روش‌های دیگری مانند اسکیماتیک (schematic entry) یا High-Level Synthesis (HLS) نیز وجود دارند که در HLS، طرح از زبان‌های سطح بالاتر مانند C/C++ به HDL ترجمه می‌شود، که فرایند طراحی را برای برنامه‌نویسان نرم‌افزار تسهیل می‌کند.

پس از ورود طرح، مرحله سنتز (Synthesis) آغاز می‌شود. در این مرحله، ابزارهای EDA (Electronic Design Automation) کد HDL را به یک Netlist از گیت‌های منطقی و

فلیپ‌فلاپ‌های استاندارد (مانند AND، OR، XOR، D-FF) ترجمه می‌کنند. این Netlist یک توصیف منطقی و خالص از مدار است که هنوز به سخت‌افزار فیزیکی خاصی در FPGA نگاشت نشده است. سپس مرحله پیاده‌سازی (Implementation) که شامل چندین زیرمرحله حیاتی است، Netlist منطقی را به یک پیکربندی فیزیکی و قابل بارگذاری برای FPGA تبدیل می‌کند. این شامل ترجمه Netlist به فرمت قابل فهم برای ابزارهای FPGA، نگاشت گیت‌های منطقی به CLBها و سایر بلوک‌های سخت‌افزاری موجود در FPGA، و سپس جانمایی و مسیریابی (Place and Route) است. در این بخش، موقعیت فیزیکی هر CLB و بلوک دیگر بر روی تراشه تعیین می‌شود و مسیرهای ارتباطی (interconnect) بین بلوک‌های جانمایی شده در شبکه اتصال‌دهنده قابل برنامه‌ریزی ایجاد می‌گردد. هدف اصلی این مرحله، دستیابی به عملکرد مورد نظر (به ویژه سرعت عملیات)، استفاده بهینه از منابع و رعایت دقیق محدودیت‌های زمانی تعیین شده است.

پس از جانمایی و مسیریابی موفقیت‌آمیز و بدون خطا، ابزار EDA یک فایل بیت‌استریم (bitstream) تولید می‌کند. این فایل در واقع یک دنباله‌ای از صفر و یک است که شامل تمام اطلاعات لازم برای پیکربندی داخلی FPGA است. این اطلاعات شامل نحوه تنظیم LUTs، فلیپ‌فلاپ‌ها و پیکربندی دقیق سوئیچ‌های شبکه اتصال‌دهنده می‌شود. در مرحله نهایی، برنامه‌ریزی FPGA (FPGA Programming) انجام می‌شود. فایل بیت‌استریم از طریق رابط‌های خاصی (مانند JTAG) به FPGA بارگذاری می‌شود. پس از بارگذاری کامل، FPGA طبق پیکربندی تعریف شده عمل می‌کند و عملکرد طراحی شده را آغاز می‌نماید.

FPGA ها به دلیل ویژگی‌های منحصربه‌فرد خود، مزایای قابل توجهی را ارائه می‌دهند که آن‌ها را در بسیاری از کاربردها به گزینه‌ای ایده‌آل و حتی ضروری تبدیل کرده است. برجسته‌ترین و شاید مهم‌ترین مزیت، قابلیت انعطاف‌پذیری و بازپیکربندی (Reconfigurability) آن پس از تولید است. این بدان معناست که می‌توان یک FPGA را بارها و بارها با طرح‌های مختلف برنامه‌ریزی کرد. این ویژگی برای توسعه و آزمایش (prototyping) بسیار ارزشمند است، زیرا به طراحان امکان می‌دهد تا بدون نیاز به ساخت مجدد سخت‌افزار، تغییرات را اعمال، خطاها را رفع و ویژگی‌های جدید را به سرعت اضافه کنند. مزیت دیگر، زمان عرضه به بازار سریع‌تر (Faster Time-to-Market) در مقایسه با ASICها است که نیاز به چرخه‌های طراحی طولانی و تولید پیچیده دارند.

علاوه بر این، FPGAها قابلیت پردازش موازی واقعی (True Parallel Processing) را فراهم می‌کنند. برخلاف پردازنده‌های نرم‌افزاری که عملیات را به صورت سریالی انجام می‌دهند، FPGAها می‌توانند چندین عملیات را به صورت کاملاً موازی و همزمان انجام دهند. این قابلیت برای کاربردهایی که نیاز به توان محاسباتی بسیار بالا دارند (مانند پردازش تصویر، رمزنگاری و هوش مصنوعی) حیاتی است. به دلیل پیاده‌سازی سخت‌افزاری توابع، FPGAها به عملکرد بالا (High Performance) دست می‌یابند و می‌توانند به فرکانس‌های عملیاتی بالاتری نسبت به پردازنده‌های عمومی برسند و عملیات را با تاخیر (latency) بسیار کمتری انجام دهند که برای سیستم‌های بلادرنگ ضروری است. در برخی موارد،

FPGAها می‌توانند در مقایسه با پردازنده‌های گرافیکی (GPUs) یا حتی پردازنده‌های مرکزی (CPUs)، مصرف انرژی کارآمدتری (Power Efficiency) داشته باشند، زیرا فقط سخت‌افزار مورد نیاز برای وظیفه خاص پیاده‌سازی می‌شود. همچنین، در کاربردهای خاص و حیاتی، FPGAها را می‌توان برای پیاده‌سازی سیستم‌های مقاوم در برابر خطا طراحی کرد که به افزایش قابلیت اطمینان و مقاومت در برابر خطا (Reliability and Fault Tolerance) کمک می‌کند.

انعطاف‌پذیری، کارایی و قابلیت برنامه‌ریزی مجدد FPGAها باعث شده است که در طیف وسیعی از صنایع و کاربردها، از جمله حوزه‌های پیشرفته و حیاتی، مورد استفاده قرار گیرند. این کاربردها شامل پردازش سیگنال دیجیتال (DSP) در مخابرات و سیستم‌های راداری، تجهیزات شبکه‌ای با سرعت بالا و ایستگاه‌های پایه بی‌سیم G۵ در مخابرات و شبکه‌سازی، تسریع‌کننده‌های سخت‌افزاری برای شبکه‌های عصبی عمیق در هوش مصنوعی و یادگیری ماشین، سیستم‌های کنترل پرواز و رادار در هوافضا و دفاع، پردازش تصویر در دستگاه‌های MRI و CT Scan در تجهیزات پزشکی، سیستم‌های کمک راننده پیشرفته (ADAS) در صنعت خودرو، پیاده‌سازی الگوریتم‌های معاملاتی با تاخیر بسیار کم در مالی و تجارت با فرکانس بالا (HFT)، و همچنین به عنوان پلتفرمی ایده‌آل برای prototyping و شبیه‌سازی ASIC قبل از تولید نهایی انبوه.

با وجود مزایای بی‌شمار، طراحی با FPGAها چالش‌های خاص خود را نیز به همراه دارد. این چالش‌ها شامل پیچیدگی طراحی و نیاز به دانش عمیق از معماری سخت‌افزار و تسلط بر زبان‌های HDL است که منحنی یادگیری نسبتاً شیب‌داری دارد. ابزارهای EDA برای FPGAها نیز پیچیده هستند و استفاده بهینه از آن‌ها نیازمند تجربه و مهارت است. همچنین، برای طرح‌های بزرگ و پیچیده، زمان سنتز و جانمایی/مسیریابی می‌تواند بسیار طولانی باشد، که می‌تواند چرخه توسعه را کند کند. در نهایت، برای تولید در مقیاس بسیار بالا و انبوه، ASICها معمولاً از نظر هزینه واحد و مصرف انرژی بهینه‌تر از FPGAها هستند. با این حال، افق آینده FPGAها بسیار روشن به نظر می‌رسد. پیشرفت‌های مداوم در تکنولوژی ساخت (استفاده از گره‌های کوچکتر)، توسعه معماری‌های جدید (مانند FPGAهای تطبیقی یا ACAPها) و ابزارهای طراحی سطح بالاتر (نظیر High-Level Synthesis - HLS) به طور پیوسته در حال بهبود قابلیت‌ها و سهولت استفاده از این تراشه‌ها هستند. نقش FPGAها در عصر محاسبات ابری، هوش مصنوعی در لبه (Edge AI) و سیستم‌های خودمختار به طور فزاینده‌ای اهمیت پیدا خواهد کرد. FPGAها نه تنها به عنوان ابزاری قدرتمند برای مهندسان الکترونیک شناخته می‌شوند، بلکه به عنوان یک پلتفرم نوآورانه برای تحقق ایده‌های خلاقانه و پیشبرد مرزهای فناوری در دنیای دیجیتال عمل می‌کنند.

ASIC، مخفف Application-Specific Integrated Circuit، یک مدار مجتمع (IC) است که به طور خاص برای یک کاربرد یا عملکرد واحد طراحی و ساخته می‌شود. برخلاف پردازنده‌های عمومی (General-Purpose Processors) یا حتی FPGAها که قابلیت پیکربندی مجدد را ارائه می‌دهند، ASICها به گونه‌ای سفارشی‌سازی می‌شوند که یک وظیفه یا مجموعه وظایف مشخص را با حداکثر کارایی،

سرعت و حداقل مصرف انرژی انجام دهند. این تراشه‌ها نمادی از اوج بهینه‌سازی سخت‌افزار هستند و نقش حیاتی در دستگاه‌های الکترونیکی روزمره ما ایفا می‌کنند، از گوشی‌های هوشمند و خودروها گرفته تا سرورهای دیتاسنتر و تجهیزات مخابراتی.

معماری یک ASIC به طور کامل به نیازهای کاربرد خاص آن وابسته است. در هسته خود، ASIC از مجموعه‌ای از گیت‌های منطقی (مانند AND، OR، NAND)، فلیپ‌فلاپ‌ها و بلوک‌های حافظه تشکیل شده است که به صورت سفارشی برای دستیابی به عملکرد مورد نظر به هم متصل شده‌اند. این طراحی کاملاً بهینه شده، به ASIC اجازه می‌دهد تا وظایف را با سرعتی بی‌سابقه و با مصرف انرژی بسیار کمتر نسبت به یک راهکار مبتنی بر نرم‌افزار یا یک تراشه قابل برنامه‌ریزی عمومی انجام دهد. هر بخش از تراشه، از آرایش گیت‌ها گرفته تا مسیرهای سیم‌کشی، با دقت فراوان برای عملکرد نهایی تنظیم می‌شود.

فرایند طراحی و تولید یک ASIC بسیار پیچیده و طولانی است و شامل چندین مرحله دقیق می‌شود. این چرخه با مشخصات (Specification) دقیق کاربرد آغاز می‌شود، جایی که تمام الزامات عملکردی، توان مصرفی و زمانی تعیین می‌گردد. سپس، در مرحله طراحی سطح بالا (High-Level Design)، معماران سیستم، ساختار کلی و بلوک‌های اصلی تراشه را تعریف می‌کنند. در ادامه، طراحی سطح گیت (Gate-Level Design) و استفاده از زبان‌های توصیف سخت‌افزار (HDL) مانند VHDL یا Verilog برای توصیف منطق مدار آغاز می‌شود. این کد HDL سپس وارد فرایند سنتز (Synthesis) می‌شود، جایی که به یک Netlist از گیت‌های استاندارد نگاشت می‌گردد. پس از آن، جانمایی (Placement) گیت‌ها و بلوک‌های منطقی بر روی تراشه انجام می‌شود، و به دنبال آن مسیریابی (Routing) دقیق اتصالات بین آن‌ها صورت می‌گیرد. این مراحل بسیار حیاتی هستند، زیرا کیفیت جانمایی و مسیریابی به طور مستقیم بر سرعت، توان مصرفی و مساحت نراشه تأثیر می‌گذارد.

پس از اتمام طراحی فیزیکی، مراحل پیچیده اعتبارسنجی (Verification) و تولید ماسک (Mask Generation) آغاز می‌شود. اعتبارسنجی شامل شبیه‌سازی‌های گسترده و آزمایش‌های عملکردی برای اطمینان از صحت طراحی و عدم وجود خطاهاست. تولید ماسک‌ها، یک فرایند بسیار گران‌قیمت و دقیق است که لایه‌های مختلف طرح را برای ساخت تراشه در کارخانه‌های نیمه‌هادی (Foundries) آماده می‌کند. در نهایت، مرحله ساخت (Fabrication) در یک کارخانه نیمه‌هادی پیشرفته انجام می‌گیرد، که شامل ده‌ها مرحله فیزیکی برای ساخت تراشه بر روی ویفرهای سیلیکونی است. پس از ساخت، تراشه‌ها تست (Testing) می‌شوند تا از عملکرد صحیح هر بخش اطمینان حاصل شود و در نهایت بسته‌بندی (Packaging) شده و برای استفاده آماده می‌گردند.

مزایای اصلی ASICها در کارایی و بهینه‌سازی آن‌ها نهفته است. عملکرد بالا (High Performance) و سرعت بسیار زیاد از ویژگی‌های بارز آن‌هاست، زیرا طراحی کاملاً سفارشی امکان دستیابی به فرکانس‌های عملیاتی بالاتر و پردازش موازی حداکثری را فراهم می‌کند. مصرف انرژی پایین (Low Power Consumption) نیز یکی دیگر از مزایای کلیدی است؛ از آنجایی که فقط منطق مورد نیاز پیاده‌سازی

می‌شود و هر بخش برای کارایی بهینه طراحی شده است، اتلاف انرژی به حداقل می‌رسد. این ویژگی برای دستگاه‌های باتری‌دار یا سیستم‌هایی که گرمای کمی تولید می‌کنند، بسیار حیاتی است. همچنین، هزینه واحد پایین (Low Unit Cost) در تولید انبوه، ASICها را به گزینه‌ای اقتصادی تبدیل می‌کند. با وجود هزینه بالای طراحی و راه‌اندازی اولیه (Non-Recurring Engineering - NRE), در حجم‌های تولید میلیونی، هزینه هر تراشه به طرز چشمگیری کاهش می‌یابد.

ASIC ها در طیف وسیعی از صنایع و کاربردها، جایی که نیاز به عملکرد بی‌نظیر و مصرف انرژی حداقل وجود دارد، به کار گرفته می‌شوند. این تراشه‌ها قلب تپنده گوشی‌های هوشمند و تبلت‌ها هستند که وظایفی مانند پردازش گرافیک، مدیریت توان، و ارتباطات بی‌سیم را انجام می‌دهند. در تجهیزات شبکه و مخابراتی، ASICها برای پردازش سریع بسته‌های داده در روترها و سوئیچ‌ها، و مدیریت سیگنال‌های رادیویی در ایستگاه‌های پایه استفاده می‌شوند. در خودروهای مدرن، ASICها در سیستم‌های پیشرفته کمک راننده (ADAS)، کنترل موتور و سیستم‌های اطلاعاتی و سرگرمی نقش دارند. در هوش مصنوعی و محاسبات با عملکرد بالا (HPC)، ASICهای تخصصی (مانند TPUهای گوگل) برای تسریع بارهای کاری یادگیری ماشین طراحی شده‌اند. همچنین، در صنایع مصرفی مانند تلویزیون‌های هوشمند، کنسول‌های بازی و لوازم خانگی نیز برای بهینه‌سازی عملکرد و کاهش هزینه از ASICها استفاده می‌شود.

با این حال، توسعه ASIC با چالش‌های قابل توجهی همراه است. هزینه‌های اولیه بالا (High NRE Costs)، که شامل هزینه‌های طراحی، ابزارها و تولید ماسک می‌شود، می‌تواند به میلیون‌ها دلار برسد. این امر ASICها را تنها برای کاربردهایی با حجم تولید بسیار بالا توجیه‌پذیر می‌سازد. زمان توسعه طولانی (Long Development Time) نیز یک چالش دیگر است؛ چرخه طراحی و تولید می‌تواند ماه‌ها یا حتی سال‌ها به طول انجامد. این موضوع باعث می‌شود که ASICها برای بازارهایی که به سرعت در حال تغییر هستند یا برای نمونه‌سازی اولیه مناسب نباشند. همچنین، عدم انعطاف‌پذیری به این معنی است که پس از تولید، عملکرد ASIC ثابت است و هر گونه تغییر یا رفع خطا مستلزم طراحی و تولید مجدد تراشه است که بسیار پرهزینه و زمان‌بر خواهد بود. با وجود این چالش‌ها، مزایای بی‌نظیر ASIC در کارایی و بهینه‌سازی، آن را به انتخابی ضروری برای کاربردهای با حجم بالا و نیازمند حداکثر عملکرد تبدیل کرده و نقش آن در پیشرفت تکنولوژی مدرن غیرقابل انکار است.

وقتی این دو فناوری را با GPU مقایسه می‌کنیم، تفاوت‌های کلیدی نمایان می‌شوند. انعطاف‌پذیری نقطه قوت FPGA است؛ این تراشه می‌تواند برای پیاده‌سازی تقریباً هر منطق دیجیتالی پیکربندی شود و در صورت نیاز، عملکرد خود را تغییر دهد. این ویژگی برای کاربردهای در حال تکامل یا نمونه‌سازی اولیه که طرح ممکن است نیاز به اصلاحات مکرر داشته باشد، بسیار ارزشمند است. در مقابل، ASICها به دلیل طراحی سفارشی خود، کمترین انعطاف‌پذیری را دارند؛ عملکرد آن‌ها پس از ساخت ثابت است. GPUها نیز در این طیف میانی قرار می‌گیرند؛ آن‌ها انعطاف‌پذیری برنامه‌نویسی نرم‌افزاری را ارائه می‌دهند، اما معماری سخت‌افزاری زیربنایی آن‌ها ثابت است و نمی‌توانند مانند FPGA منطق گیت‌ها را تغییر دهند.

از نظر عملکرد و سرعت، ASICها در وظیفه خاص خود، بالاترین کارایی مطلق را ارائه می‌دهند، زیرا هر گیت برای آن وظیفه بهینه شده است. GPUها در کارهای با موازی‌سازی داده بالا (مانند آموزش شبکه‌های عصبی) عملکرد بسیار قدرتمندی دارند، اما ممکن است برای کارهای بلادرنگ با تاخیر بسیار پایین به اندازه FPGA بهینه نباشند. FPGAها می‌توانند برای دستیابی به تاخیر بسیار پایین (Low Latency) و توان عملیاتی بالا (High Throughput) در وظایف خاص، به خصوص در پردازش‌های جریان داده، بهینه‌سازی شوند، اما دستیابی به حداکثر عملکرد آن‌ها نیازمند تخصص بالای سخت‌افزاری است. در زمینه مصرف انرژی، ASICها معمولاً بهینه‌ترین هستند، زیرا فقط سخت‌افزار مورد نیاز فعال است. FPGAها نیز می‌توانند بسیار کارآمد باشند، به خصوص در کاربردهای لبه‌ای که توان مصرفی محدود است، اما به طور کلی ممکن است از ASICها انرژی بیشتری مصرف کنند. GPUها، به دلیل قدرت پردازشی عظیم خود، معمولاً بیشترین مصرف انرژی را در میان این سه دارند، به خصوص در اوج بار کاری.

هزینه نیز یک عامل تعیین‌کننده است. ASICها بالاترین هزینه اولیه توسعه (NRE) را دارند که می‌تواند به میلیون‌ها دلار برسد، اما در تولید انبوه، هزینه واحد آن‌ها به پایین‌ترین حد می‌رسد. FPGAها دارای هزینه اولیه متوسط (گران‌تر از GPUهای مصرفی، اما بسیار ارزان‌تر از NRE یک ASIC) و هزینه واحد بالاتر از ASIC در تولید انبوه هستند. GPUها معمولاً پایین‌ترین هزینه اولیه توسعه (برای شروع با یک کارت گرافیک استاندارد) و هزینه واحد نسبتاً پایین‌تری نسبت به FPGA دارند، که آن‌ها را برای تحقیقات و توسعه گسترده در هوش مصنوعی بسیار در دسترس قرار داده است.

در حوزه هوش مصنوعی، این سه فناوری نقش‌های مکمل یکدیگر را ایفا می‌کنند. GPUها به دلیل توانایی بی‌نظیرشان در پردازش موازی، به پادشاهان بلامنازع آموزش مدل‌های هوش مصنوعی (AI Training) تبدیل شده‌اند، جایی که حجم عظیمی از داده‌ها و محاسبات فلوتینگ‌پوینت مورد نیاز است. اما در فاز استنتاج (AI Inference)، که مدل آموزش‌دیده برای پیش‌بینی و انجام وظیفه استفاده می‌شود، گزینه‌ها متنوع‌تر می‌شوند. FPGAها با قابلیت تاخیر پایین و کارایی انرژی بالا، برای استنتاج بلادرنگ در لبه شبکه یا سیستم‌های تعبیه‌شده (Embedded Systems) بسیار مناسب هستند. ASICهای هوش مصنوعی (مانند TPUهای گوگل) نیز برای استنتاج در مقیاس بسیار بزرگ و با حداکثر بهینه‌سازی در دیتاسنترها یا کاربردهای خاص (مانند پردازش گفتار در یک گوشی هوشمند) به کار می‌روند. به طور خلاصه، GPUها برای آموزش مدل‌های هوش مصنوعی، FPGAها برای استنتاج در لبه و ASICها برای استنتاج در مقیاس بزرگ یا در دستگاه‌های مصرفی، هر یک جایگاه منحصر به فرد خود را دارند. این تنوع در شتاب‌دهنده‌های سخت‌افزاری نشان‌دهنده نیاز روزافزون به قدرت پردازشی تخصصی و بهینه برای پیشبرد مرزهای فناوری در عصر دیجیتال است.

فصل چهارم: برنامه‌نویسی GPU

برنامه‌نویسی واحد پردازش گرافیکی (GPU) به فرآیند توسعه و بهینه‌سازی کد برای بهره‌برداری از توان پردازش موازی GPUها اشاره دارد. این رویکرد امکان انجام عملیات محاسباتی عمومی با کارایی بسیار بالا را به صورت همزمان فراهم می‌آورد. در گذشته، GPUها عمدتاً برای رندرینگ گرافیک کامپیوتری استفاده می‌شدند، اما امروزه کاربرد آنها به طیف وسیعی از وظایف محاسباتی عمومی گسترش یافته است. طراحی ذاتی GPU به آن اجازه می‌دهد تا عملیات مشابه را بر روی مقادیر زیادی از داده‌ها به صورت موازی انجام دهد، که این ویژگی کارایی پردازش را برای بسیاری از وظایف محاسباتی فشرده به طرز چشمگیری افزایش می‌دهد.

اهمیت برنامه‌نویسی GPU در عصر محاسبات مدرن، به دلیل توانایی آن در تسریع چشمگیر بارهای کاری پیچیده و داده‌محور، رو به فزونی است. این فناوری در حوزه‌هایی نظیر هوش مصنوعی (AI)، یادگیری ماشین (ML)، بازی‌های ویدئویی، شبیه‌سازی‌های علمی، و تجزیه و تحلیل داده‌های بزرگ نقش محوری ایفا می‌کند. تأثیر تحول‌آفرین برنامه‌نویسی GPU در صنایع مختلفی همچون مالی (بهبود محاسبات ریسک)، سرگرمی (جلوه‌های بصری خیره‌کننده در فیلم‌ها) و تصویربرداری پزشکی (اسکن‌های MRI و CT سریع‌تر و واضح‌تر) مشهود است. مزایای کلیدی برنامه‌نویسی GPU شامل سرعت بالای پردازش، بهره‌وری انرژی (پردازش محاسبات بیشتر در ازای هر وات مصرفی)، مقرون به صرفه بودن (عملکرد بهتر به ازای هر دلار در مقایسه با CPU برای بارهای کاری مناسب)، و مقیاس‌پذیری (افزودن GPUهای بیشتر برای مدیریت بارهای کاری بزرگتر) است. این قابلیت‌ها، برنامه‌نویسی GPU را به یک فناوری تحول‌آفرین برای دستیابی به امکانات پیشگامانه در محاسبات با کارایی بالا تبدیل کرده است.

برنامه‌نویسی برای GPUها نیازمند یک الگو و چارچوب متفاوت است که بر محاسبات موازی تمرکز دارد. GPUها در وظایفی که می‌توانند به عملیات کوچکتر، مستقل و تکراری تقسیم شوند، برتری دارند. این امر مستلزم برنامه‌نویسی موازی صریح است که از مدل‌های مختلفی برای بهره‌برداری از هزاران هسته GPU استفاده می‌کند.

NVIDIA CUDA

همانگونه که قبلا گفته شد CUDA پلتفرم محاسبات موازی و مدل API اختصاصی توسعه یافته توسط NVIDIA است. این پلتفرم به توسعه‌دهندگان امکان می‌دهد تا با استفاده از زبان‌های محبوبی مانند C،

C++، Fortran، Python، Julia و MATLAB، با افزودن افزونه‌ها و کلمات کلیدی، از قدرت GPUها برای محاسبات عمومی بهره ببرند. CUDA بر اساس یک معماری موازی گسترده عمل می‌کند که از هزاران رشته پشتیبانی می‌کند.

انتزاعات کلیدی در مدل برنامه‌نویسی CUDA شامل سلسله مراتب گروه‌های رشته (شبکه‌ها، بلوک‌ها و رشته‌ها)، سلسله مراتب حافظه‌ها (حافظه محلی برای هر رشته، حافظه مشترک برای رشته‌های یک بلوک، و حافظه سراسری برای تمام رشته‌ها) و مکانیزم‌های همگام‌سازی مانند موانع است. جریان کاری معمول در برنامه‌نویسی CUDA شامل بارگذاری داده‌ها در حافظه CPU، کپی داده‌ها از CPU به حافظه GPU، فراخوانی هسته GPU، کپی نتایج از GPU به CPU و سپس استفاده از نتایج روی CPU است. CUDA می‌تواند سرعت برخی برنامه‌ها را ۳۰ تا ۱۰۰ برابر افزایش دهد و از مزایایی مانند حافظه یکپارچه و مجازی، حافظه مشترک با سرعت بالا و پشتیبانی کامل از عملیات بیتی و صحیح بهره می‌برد. با این حال، ماهیت اختصاصی آن و قابلیت همکاری یک‌طرفه با زبان‌های دیگر از محدودیت‌های آن محسوب می‌شود.

مفهوم Warp در معماری CUDA

در معماری پردازنده‌های گرافیکی شرکت NVIDIA، به‌ویژه آن‌هایی که از CUDA استفاده می‌کنند، یک مفهوم کلیدی و مهم به نام Warp وجود دارد. Warp در ساده‌ترین تعریف، یک واحد اجرای گروهی از ۳۲ نخ (Thread) است که به‌صورت هم‌زمان و هماهنگ توسط یک Streaming Multiprocessor (SM) اجرا می‌شوند. در واقع، وقتی برنامه‌نویس در CUDA کدی می‌نویسد که شامل هزاران نخ است، این نخ‌ها پشت‌صحنه به صورت دسته‌هایی ۳۲ تایی گروه‌بندی می‌شوند و هر دسته، یک warp را تشکیل می‌دهد. این ساختار کمک می‌کند تا GPU بتواند از معماری SIMD (Single Instruction, Multiple Data) به‌شکل مؤثری استفاده کند، یعنی یک دستور واحد را روی چند داده مختلف اعمال کند.

در هر لحظه، تمام نخ‌های یک warp دستور یکسانی را اجرا می‌کنند، ولی ممکن است روی داده‌های متفاوتی کار کنند. این بدان معناست که اگر نخ‌های مختلف داخل یک warp شاخه‌های متفاوتی از برنامه را دنبال کنند (مثلاً در یک if شرطی بعضی اجرا شوند و بعضی نه)، پدیده‌ای به نام divergence (انشعاب درون‌وارپی) رخ می‌دهد. در چنین حالتی، warp باید مسیرهای مختلف را به نوبت اجرا کند و این منجر به کاهش کارایی می‌شود، چرا که بخشی از نخ‌ها در زمان اجرای مسیرهای دیگر غیرفعال باقی می‌مانند. بنابراین، برای بهره‌برداری بهینه از GPU، برنامه‌نویسان باید سعی کنند نخ‌های درون یک warp را طوری طراحی کنند که مسیرهای اجرای مشابهی را دنبال کنند.

از آنجایی که مدیریت warpها در سطح سخت‌افزاری انجام می‌شود، برنامه‌نویس معمولاً مستقیماً با مفهوم warp سروکار ندارد، اما درک آن برای بهینه‌سازی عملکرد بسیار حیاتی است. مثلاً اگر برنامه‌ای طوری نوشته شود که نخ‌های هر warp دسترسی به حافظه‌های متوالی داشته باشند (به‌جای تصادفی)،

عملکرد حافظه بسیار بهبود می‌یابد چون GPU می‌تواند آن‌ها را در قالب memory coalescing مدیریت کند. همچنین، الگوریتم‌هایی که به‌طور طبیعی «موازی و بدون انشعاب» هستند، در اجرای درون‌وارپی بازده بالاتری خواهند داشت. در نتیجه، Warp یکی از عناصر بنیادی در طراحی، تحلیل و بهینه‌سازی برنامه‌های CUDA محسوب می‌شود.

CUDA C

CUDA C یک گسترش از زبان برنامه‌نویسی C/C++ است که توسط شرکت NVIDIA ارائه شده و به برنامه‌نویسان اجازه می‌دهد تا از توان پردازشی بالای کارت‌های گرافیک NVIDIA برای انجام محاسبات موازی استفاده کنند. واژه CUDA مخفف Compute Unified Device Architecture است و این پلتفرم به گونه‌ای طراحی شده که امکان برنامه‌نویسی برای GPU را همانند برنامه‌نویسی برای CPU ساده‌سازی کند. در CUDA C، توسعه‌دهنده می‌تواند بخشی از کد خود را به صورت کرنل (Kernel) بنویسد؛ این کرنل‌ها روی هزاران نخ (Thread) در GPU به صورت هم‌زمان اجرا می‌شوند، که برای پردازش‌های سنگین مانند یادگیری ماشین، شبیه‌سازی علمی، پردازش تصویر و ویدیو بسیار مناسب است.

در CUDA C، کد برنامه به دو بخش اصلی تقسیم می‌شود: کدی که روی CPU (میزبان یا host) اجرا می‌شود و کدی که روی GPU (دستگاه یا device) اجرا می‌شود.

در CUDA، کرنل (Kernel) در واقع تابعی است که روی GPU اجرا می‌شود و به طور هم‌زمان توسط تعداد زیادی نخ (thread) اجرا می‌گردد. این تابع با کلمه‌ی کلیدی __global__ تعریف می‌شود تا مشخص شود که این تابع توسط CPU فراخوانی شده ولی بر روی GPU اجرا می‌شود. برخلاف توابع عادی در C/C++، کرنل‌ها نمی‌توانند مقدار بازگشتی داشته باشند (یعنی نوع void دارند) و برای دریافت نتایج باید از حافظه‌ای که به اشتراک گذاشته شده یا از طریق کپی بین حافظه‌ها استفاده کرد. این کرنل‌ها به وسیله دستور ویژه‌ای مانند

```
kernel<<<blocks, threads>>>
```

از سمت CPU اجرا می‌شوند. همچنین، CUDA امکانات گسترده‌ای برای مدیریت حافظه بین CPU و GPU ارائه می‌دهد، از جمله تخصیص حافظه، کپی داده، و آزادسازی منابع. به عنوان مثال تابع `cudaMalloc` برای رزرو کردن حافظه‌ای روی GPU به کار می‌رود. تابع `cudaMemcpy` داده را بین CPU و GPU جابجا می‌کند و یا `cudaFree` حافظه‌ای که روی GPU رزرو شده بود را آزاد می‌کند.

همچنین برای برنامه‌نویسی با CUDA C، لازم است CUDA Toolkit را از سایت رسمی NVIDIA نصب کنید. این مجموعه شامل کامپایلر nvcc، کتابخانه‌ها، ابزارهای اشکال‌زدایی و نمونه کدها است.

به عنوان نمونه این قطعه کد که با CUDA C نوشته شده عدد ۵ را به تمام عناصر یک آرایه اضافه میکند

```
#include <iostream>

__global__ void addFive(int *data) {
    int idx = threadIdx.x;
    data[idx] += 5;
}

int main() {
    const int N = 5;
    int h_data[N] = {1, 2, 3, 4, 5};

    int *d_data;
    cudaMalloc(&d_data, N * sizeof(int));
    cudaMemcpy(d_data, h_data, N * sizeof(int), cudaMemcpyHostToDevice);

    addFive<<<1, N>>>(d_data);

    cudaMemcpy(h_data, d_data, N * sizeof(int), cudaMemcpyDeviceToHost);
    cudaFree(d_data);

    std::cout << "result: ";
    for (int i = 0; i < N; ++i)
        std::cout << h_data[i] << " ";
    std::cout << std::endl;

    return 0;
}
```

در این کد، یک آرایه ۵ عنصری در حافظه CPU تعریف می‌شود. این آرایه به GPU منتقل شده و روی GPU یک کرنل ساده اجرا می‌شود که عدد ۵ را به هر عنصر اضافه کند. نتیجه از GPU به CPU برگردانده شده و چاپ می‌شود.

در CUDA، کرنل (Kernel) در واقع تابعی است که روی GPU اجرا می‌شود و به طور هم‌زمان توسط تعداد زیادی نخ (thread) اجرا می‌گردد. این تابع با کلمه‌ی کلیدی __global__ تعریف می‌شود تا مشخص شود که این تابع توسط CPU فراخوانی شده ولی بر روی GPU اجرا می‌شود. برخلاف توابع عادی در C/C++، کرنل‌ها نمی‌توانند مقدار بازگشتی داشته باشند (یعنی نوع void دارند) و برای دریافت نتایج باید از حافظه‌ای که به اشتراک گذاشته شده یا از طریق کپی بین حافظه‌ها استفاده کرد.

استفاده CUDA در پایتون

پایتون به خودی خود از پردازش موازی GPU پشتیبانی نمی‌کند، اما این کتابخانه‌ها امکان دسترسی به GPU را فراهم می‌کنند و به کاربر اجازه می‌دهند تا توابعی را بنویسد که مستقیماً روی GPU اجرا شوند. یکی از رایج‌ترین ابزارها برای این کار Numba است که توسط Anaconda توسعه داده شده و با استفاده از دکوراتورها، کدهای پایتونی را به کد CUDA کامپایل می‌کند. Numba به برنامه‌نویسان اجازه می‌دهد که بدون نیاز به نوشتن کد C، از قدرت CUDA استفاده کنند.

برای استفاده از CUDA با Numba، باید اطمینان حاصل کرد که درایورهای NVIDIA و CUDA Toolkit به‌درستی نصب شده‌اند. در Numba، می‌توان با دکوراتور `@cuda.jit` توابع کرنل تعریف کرد. این توابع همانند کرنل‌های CUDA C تعریف می‌شوند، ولی با سینتکس پایتون. داده‌ها باید به شکل آرایه‌های NumPy باشند، که به حافظه GPU منتقل می‌شوند. همچنین می‌توان با استفاده از متغیرهای `cuda.blockIdx` , `cuda.threadIdx` و غیره، موقعیت هر نخ را مشخص کرد

همان مثالی که در محیط C/C++ نوشته بودیم را برای python هم مینویسیم که برنامه عدد ۵ را به تمام عناصر یک آرایه اضافه کند

```
import numpy as np
from numba import cuda

@cuda.jit
def add_five_kernel(arr):
    idx = cuda.grid(1)
    if idx < arr.size:
        arr[idx] += 5

n = 10
arr = np.arange(n, dtype=np.int32)
d_arr = cuda.to_device(arr)
threads_per_block = 32
blocks_per_grid = (n + (threads_per_block - 1)) // threads_per_block
add_five_kernel[blocks_per_grid, threads_per_block](d_arr)
result = d_arr.copy_to_host()
print("Result:", result)
```

در این کد یک کرنل به نام `add_five_kernel` تعریف شده که به هر عنصر آرایه عدد ۵ اضافه می‌کند. داده‌ها از طریق تابع `cuda.to_device` به حافظه GPU منتقل می‌شوند. کرنل با مشخص کردن تعداد بلاک و نخ در هر بلاک اجرا می‌شود. نتیجه با تابع `copy_to_host` به حافظه اصلی بازگردانده می‌شود.

PyTorch

کتابخانه‌ی PyTorch یکی از محبوب‌ترین ابزارهای یادگیری ماشین در پایتون است که توسط Facebook توسعه داده شده و از GPU برای شتاب‌دهی به محاسبات استفاده می‌کند. استفاده از CUDA در PyTorch نسبت به CUDA خام یا Numba بسیار ساده‌تر و سطح‌بالاتر است، چرا که PyTorch همه‌ی عملیات حافظه، پردازش و مدیریت دستگاه را با چند دستور ساده کنترل می‌کند. برنامه‌نویس به جای آنکه با کرنل‌ها و نخ‌ها و حافظه‌ی دستگاه سر و کله بزند، فقط با اشیای Tensor کار می‌کند و با انتقال آن‌ها به GPU، عملیات به‌طور خودکار روی GPU انجام می‌شود.

یک مثال ساده :

```
import torch

if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

a = torch.arange(10, dtype=torch.float32).to(device)
b = torch.ones(10, dtype=torch.float32).to(device)
result = a + b
print("Result:", result)
```

در این کد ابتدا بررسی می‌شود که آیا CUDA در دسترس است یا نه. دو تنسور(ساختار اصلی داده در PyTorch) روی GPU تعریف می‌شوند. عملیات جمع به‌طور کامل روی GPU انجام می‌شود. در پایان، نتیجه روی همان دستگاه باقی می‌ماند مگر آنکه به cpu منتقل شود.

PyTorch از این نظر بسیار قدرتمند است که بدون نیاز به نوشتن کد CUDA، محاسبات سنگین را روی GPU اجرا می‌کند. حتی آموزش مدل‌های شبکه‌های عصبی با استفاده از CUDA تنها با یک خط انتقال مدل به GPU انجام می‌شود.

AMD ROCm

ROCm مخفف Radeon Open Compute است و یک پلتفرم متن‌باز است که توسط AMD توسعه یافته تا دسترسی به محاسبات سطح پایین روی کارت‌های گرافیک AMD را برای توسعه‌دهندگان فراهم کند. ROCm شامل ابزارها، درایورها، کتابخانه‌ها و APIهایی است که برای ایجاد، مدیریت و اجرای کدهای محاسباتی روی GPU طراحی شده‌اند. این پلتفرم به طور خاص برای سیستم‌های لینوکسی طراحی شده و بر اساس استانداردهای باز مانند OpenCL و LLVM ساخته شده است.

در مقایسه با CUDA که اختصاصاً برای کارت‌های NVIDIA طراحی شده است، ROCm از معماری باز استفاده می‌کند و امکان استفاده از GPUهای مختلف را فراهم می‌سازد. همچنین، ROCm از زبان‌های سطح بالا مانند Python و C++ نیز پشتیبانی می‌کند و رابط‌هایی برای کتابخانه‌های معروف مانند TensorFlow، PyTorch و ONNX Runtime ارائه داده است.

در دل اکوسیستم ROCm، زبان برنامه‌نویسی HIP (مخفف Heterogeneous-compute Interface for Portability) قرار دارد. HIP زبانی است که شباهت بسیار زیادی به CUDA دارد و به توسعه‌دهندگان اجازه می‌دهد کدی بنویسند که هم روی کارت‌های AMD و هم NVIDIA قابل اجرا باشد (با اندکی تغییر یا حتی بدون تغییر). HIP در واقع یک لایه تطبیق (abstraction layer) بین کد CUDA و ROCm است. با استفاده از ابزار hipify, کدهای CUDA می‌توانند به HIP تبدیل شوند.

نوشتن کد HIP از لحاظ سینتکس بسیار شبیه به CUDA C است. به‌عنوان مثال، فراخوانی کرنل‌ها همچنان با سینتکس `kernel<<<blocks, threads>>>` انجام می‌شود. HIP به توسعه‌دهندگان اجازه می‌دهد تا کدهای پرکاربرد CUDA را بدون بازنویسی کامل برای پلتفرم AMD نیز قابل اجرا کنند، که این مسأله مزیت بزرگی در سازگاری بین‌پلتفرمی محسوب می‌شود.

معماری ROCm به شکل ماژولار طراحی شده و این امکان را می‌دهد که توسعه‌دهندگان بتوانند مؤلفه‌های مورد نیاز خود را جداگانه نصب یا سفارشی‌سازی کنند. این معماری شامل مؤلفه‌هایی مانند درایور ROCk (ROCm Kernel Driver)، کامپایلرهای HSAIL و LLVM-based، ابزارهای بررسی کارایی مانند rocprof و rocr، و محیط‌های زمان اجرای مختلف برای برنامه‌نویسی موازی است. یکی از مؤلفه‌های مهم دیگر در این مجموعه ROCr Runtime است که نقش زمان اجرای اصلی را ایفا می‌کند و با APIهای سطح پایین برای مدیریت حافظه، زمان‌بندی کرنل و هماهنگی میان CPU و GPU کار می‌کند.

همچنین، ROCm در حال حاضر به شکل فعال توسط AMD و جامعه‌ی منبع‌باز توسعه داده می‌شود و از مدل مشارکتی بهره می‌برد. این یعنی توسعه‌دهندگان می‌توانند کد منبع ROCm را بررسی کرده، بهبود دهند یا به صورت محلی برای پلتفرم‌های خاص خود سفارشی‌سازی کنند. در محیط‌هایی مانند HPC (High-Performance Computing) که وابستگی به عملکرد بالا و کنترل دقیق بر منابع وجود دارد،

این نوع شفافیت و دسترسی پایین‌سطحی بسیار مهم است. به همین دلیل ROCm به‌ویژه در مراکز داده، ابررایانه‌ها و پروژه‌های علمی که نیاز به بیشینه‌سازی عملکرد دارند، مورد توجه قرار گرفته است.

در زیر یک نمونه کد HIP C که یک رقم را روی GPU افزایش می‌دهد. آورده شده

```
#include <hip/hip_runtime.h>
#include <iostream>

__global__ void increment(int *result) {
    *result += 1;
}

int main() {
    int *d_result, result = 10;

    hipMalloc(&d_result, sizeof(int));
    hipMemcpy(d_result, &result, sizeof(int), hipMemcpyHostToDevice);

    hipLaunchKernelGGL(increment, dim3(1), dim3(1), 0, 0, d_result);

    hipMemcpy(&result, d_result, sizeof(int), hipMemcpyDeviceToHost);

    std::cout << result << std::endl;

    hipFree(d_result);
    return 0;
}
```

در این کد کرنل `increment` فقط یک عدد را در مکان حافظه GPU افزایش می‌دهد. از تابع `hipLaunchKernelGGL` برای اجرای کرنل استفاده می‌شود. فضایی در حافظه GPU با تابع `hipMalloc` رزرو شده و با تابع `hipMemcpy` انتقال داده می‌شود.

OpenACC

OpenACC یک استاندارد برنامه‌نویسی موازی است که به دانشمندان و مهندسان امکان می‌دهد تا کد خود را برای شتاب‌دهنده‌های محاسباتی مانند پردازنده‌های گرافیکی (GPUs) و سایر پردازنده‌های موازی به راحتی بهینه‌سازی کنند، بدون اینکه نیاز به یادگیری پیچیدگی‌های برنامه‌نویسی موازی سطح پایین داشته باشند. این استاندارد بر پایه دستورالعمل‌ها (directives) استوار است که به کامپایلر اعلام می‌کنند کدام بخش‌های کد باید به صورت موازی اجرا شوند. کامپایلر سپس مسئولیت تولید کد موازی برای دستگاه هدف را بر عهده می‌گیرد، که این امر فرآیند برنامه‌نویسی را به شدت ساده می‌کند و به کاربران اجازه می‌دهد تا روی منطق برنامه خود تمرکز کنند.

یکی از مزایای کلیدی OpenACC قابل حمل بودن آن است. کدی که با استفاده از دستورالعمل‌های OpenACC نوشته شده، می‌تواند بر روی انواع مختلفی از شتاب‌دهنده‌ها اجرا شود، به شرطی که کامپایلر مربوطه از OpenACC پشتیبانی کند. این ویژگی باعث می‌شود که برنامه‌نویسان نگران جزئیات خاص سخت‌افزار نباشند و کد آن‌ها با تغییر سخت‌افزار همچنان قابل استفاده و کارآمد باقی بماند. OpenACC ابزاری قدرتمند برای تسریع برنامه‌های محاسباتی سنگین است که به طور معمول در زمینه‌هایی مانند شبیه‌سازی‌های علمی، تحلیل داده‌ها و یادگیری ماشین کاربرد دارند.

OpenACC به برنامه‌نویسان امکان می‌دهد تا به تدریج برنامه خود را موازی‌سازی کنند. این رویکرد تدریجی به این معناست که می‌توانند بخش‌های کوچک‌تر و محاسباتی‌تر کد را شناسایی کرده و با اضافه کردن چند دستورالعمل OpenACC، آن بخش‌ها را برای اجرا بر روی شتاب‌دهنده آماده کنند. نیازی به بازنویسی کامل برنامه نیست، که این موضوع زمان و تلاش لازم برای موازی‌سازی را به حداقل می‌رساند. این قابلیت به خصوص برای کدهای میراثی (legacy codes) که سال‌هاست توسعه یافته‌اند و بازنویسی کامل آن‌ها دشوار است، بسیار مفید است.

دستورالعمل‌های OpenACC بسیار شبیه به کامنت‌ها هستند و در صورت عدم پشتیبانی کامپایلر از OpenACC، توسط آن نادیده گرفته می‌شوند. این ویژگی باعث می‌شود که کد OpenACC همچنان بتواند توسط کامپایلرهای استاندارد پردازش شود و بر روی CPUهای معمولی اجرا گردد. این "fallback" مکانیسم بسیار مهم است زیرا تضمین می‌کند که برنامه‌ها حتی در محیط‌هایی که شتاب‌دهنده در دسترس نیست، همچنان قابل اجرا باقی بمانند و این انعطاف‌پذیری OpenACC را افزایش می‌دهد.

در نهایت، OpenACC یک راه حل کارآمد و نسبتاً آسان برای بهره‌برداری از قدرت شتاب‌دهنده‌های محاسباتی است. این استاندارد با کاهش پیچیدگی‌های برنامه‌نویسی موازی و ارائه یک مدل برنامه‌نویسی مبتنی بر دستورالعمل، به دانشمندان و مهندسان کمک می‌کند تا به سرعت و به طور موثر از سخت‌افزارهای موازی برای حل مسائل پیچیده استفاده کنند. جامعه رو به رشد کاربران و پشتیبانی از سوی فروشندگان

سخت‌افزار و نرم‌افزار، OpenACC را به یک انتخاب جذاب برای برنامه‌نویسی با کارایی بالا تبدیل کرده است.

یک نمونه ساده از کد OpenACC برای موازی‌سازی یک حلقه‌ی جمع‌آوری عناصر یک آرایه آورده شده است.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int N = 1000000;
    float sum = 0.0f;
    float *a = (float*)malloc(N * sizeof(float));

    for (int i = 0; i < N; i++) {
        a[i] = 1.0f;
    }

    #pragma acc parallel loop reduction(+:sum)
    for (int i = 0; i < N; i++) {
        sum += a[i];
    }

    printf("Sum = %f\n", sum);

    free(a);
    return 0;
}
```

در این کد یک آرایه a از نوع float به اندازه N ساخته شده و تمام عناصر آن برابر ۱ شده‌اند.

`#pragma acc parallel loop reduction(+:sum)`

به کامپایلر می‌گوید که حلقه‌ی بعدی را به صورت موازی روی سخت‌افزار شتاب‌دهنده (مثلاً GPU) اجرا کند.

`reduction(+:sum)`

یعنی متغیر sum به صورت تجمعی به‌روزرسانی شود تا از مشکلات رقابت (race condition) جلوگیری شود.

(Open Computing Language) OpenCL

OpenCL یک چارچوب برنامه‌نویسی باز که برای نوشتن برنامه‌هایی که روی پلتفرم‌های ناهمگن شامل CPU ، GPU ، واحدهای پردازش دیجیتال (DSPs) و سایر شتاب‌دهنده‌های محاسباتی استفاده می‌شود. هدف اصلی OpenCL ارائه یک استاندارد جامع برای محاسبات موازی روی انواع مختلف سخت‌افزارها است، به طوری که برنامه‌نویسان بتوانند کد خود را یک بار بنویسند و آن را بر روی پلتفرم‌های مختلف اجرا کنند. این امر OpenCL را به یک ابزار قدرتمند برای برنامه‌هایی با نیازهای محاسباتی بالا در حوزه‌هایی مانند یادگیری ماشین، پردازش تصویر، شبیه‌سازی‌های علمی و مالی تبدیل کرده است.

برخلاف OpenACC که بیشتر بر پایه دستورالعمل‌های کامپایلر است، OpenCL یک API (رابط برنامه‌نویسی کاربردی) سطح پایین‌تر را فراهم می‌کند. این API به برنامه‌نویسان کنترل دقیق‌تری بر روی مدیریت حافظه، زمان‌بندی کارها و ارتباط بین هاست (CPU) و دستگاه‌های محاسباتی (مانند GPU) می‌دهد.

معماری OpenCL شامل چندین جزء اصلی است: یک مدل پلتفرم، یک مدل حافظه، یک مدل اجرا و یک زبان برای نوشتن کرنل‌ها (OpenCL C). مدل پلتفرم نحوه شناسایی دستگاه‌های محاسباتی موجود در یک سیستم را تعریف می‌کند. مدل حافظه سلسله مراتب حافظه را در دستگاه‌های OpenCL (مانند حافظه گلوبال، لوکال، خصوصی) توصیف می‌کند. مدل اجرا نحوه توزیع و اجرای کرنل‌ها بر روی دستگاه‌ها را مدیریت می‌کند. و OpenCL C، که بر پایه استاندارد C99 است، به برنامه‌نویسان امکان می‌دهد تا کدی بنویسند که به صورت موازی روی هزاران هسته پردازشی اجرا شود.

قابل حمل بودن یکی از نقاط قوت اصلی OpenCL است. برنامه‌ای که با OpenCL نوشته شده، می‌تواند بر روی هر سخت‌افزار سازگار با OpenCL، بدون نیاز به بازنویسی مجدد، اجرا شود. این ویژگی برای توسعه‌دهندگان بسیار با ارزش است زیرا به آن‌ها امکان می‌دهد تا از یک پایگاه کد برای طیف وسیعی از دستگاه‌ها و فروشندگان سخت‌افزار استفاده کنند. این امر باعث کاهش زمان توسعه و افزایش دسترسی به بازارهای مختلف می‌شود و به استاندارد شدن محاسبات ناهمگن کمک می‌کند.

با وجود قدرت و انعطاف‌پذیری، منحنی یادگیری OpenCL به دلیل سطح پایین‌تر آن، می‌تواند شیب‌دارتر از سایر فریم‌ورک‌های موازی‌سازی مانند OpenACC یا OpenMP باشد. برنامه‌نویسان نیاز دارند که مفاهیم پیچیده‌تری مانند مدیریت صف دستورات، کنترل رویدادها، و همگام‌سازی بین وظایف را درک کنند. با این حال، برای کاربردهایی که نیاز به حداکثر عملکرد و کنترل دقیق بر روی سخت‌افزار دارند، OpenCL یک انتخاب قدرتمند و بهینه است که امکان بهره‌برداری کامل از قابلیت‌های محاسباتی دستگاه‌های مختلف را فراهم می‌کند.

فصل پنجم: کاربردهای GPU در دیگر شاخه ها

گرافیک رایانه‌ای و صنعت gaming

گرافیک رایانه‌ای (Computer Graphics) یکی از حوزه‌های بنیادین و در عین حال پویای علوم رایانه است که هدف آن، تولید و نمایش تصاویر بصری با استفاده از رایانه می‌باشد. از نخستین بازی‌های ساده‌ی پیکسلی دهه ۱۹۷۰ تا بازی‌های واقع‌گرایانه‌ی امروزی، پیشرفت در سخت‌افزارهای گرافیکی، به‌ویژه پردازنده‌های گرافیکی (GPU)، نقشی محوری در تحول کیفیت گرافیکی و پاسخ‌گویی بلادرنگ بازی‌ها داشته است.

GPU ها با قدرت محاسباتی بالا و معماری مبتنی بر موازی‌سازی، وظایف سنگین پردازش تصویر را به‌عهده می‌گیرند و به طراحان بازی این امکان را می‌دهند تا دنیایی پیچیده، سه‌بعدی و غنی از جزئیات خلق کنند که در گذشته حتی تصور آن نیز دشوار بود. امروزه، صنعت بازی به یک صنعت چندمیلیارد دلاری بدل شده است و یکی از محرک‌های اصلی پیشرفت فناوری GPU به‌شمار می‌رود.

در بازی‌های رایانه‌ای، هر آن‌چه که کاربر روی صفحه نمایش می‌بیند، از شخصیت‌ها و مناظر گرفته تا سایه‌ها و نورها، باید به‌صورت بلادرنگ (Real-time) توسط سیستم محاسبه و تولید شود. این فرآیند شامل چندین مرحله پیچیده است:

۱. مدل‌سازی سه‌بعدی (۳D Modeling):

فرآیند مدل‌سازی سه‌بعدی پایه‌ی اصلی ساخت دنیای مجازی در بازی‌های رایانه‌ای است. در این مرحله، تمامی اشیاء، شخصیت‌ها، مناظر، سازه‌ها و عناصر فیزیکی محیط با استفاده از هندسه‌ی چندضلعی (معمولاً مثلث‌ها یا چهارضلعی‌ها) ساخته می‌شوند. هر مدل از مجموعه‌ای از نقاط (Vertices)، لبه‌ها (Edges) و سطوح (Faces) تشکیل شده که ساختار مش (Mesh) را تشکیل می‌دهند.

مدلسازان حرفه‌ای معمولاً از نرم‌افزارهایی مانند Blender، Maya، یا ۳ds Max استفاده می‌کنند تا اشکال پیچیده و ارگانیک خلق کنند. گاهی مدل‌سازی به‌صورت دستی صورت می‌گیرد و گاهی از طریق اسکن سه‌بعدی یا تکنیک‌های فوتوگرامتری انجام می‌شود. مدل‌ها معمولاً در حالت "low-poly" (تعداد کم چندضلعی‌ها) طراحی می‌شوند تا کارایی و سرعت در بازی‌های بلادرنگ حفظ شود، اما در صورت نیاز با تکنیک‌هایی مانند نرمال‌مپ‌سازی (Normal Mapping) به نظر می‌رسند که بسیار دقیق و پرجزئیات‌اند.

۲. نورپردازی (Lighting):

نورپردازی در بازی‌های رایانه‌ای نه‌تنها برای افزایش واقع‌گرایی اهمیت دارد، بلکه در هدایت توجه بازیکن، ایجاد حال‌وهوای محیط، و حتی پیش‌برد داستان نقش کلیدی ایفا می‌کند. در این مرحله، سیستم باید مشخص کند که منابع نوری (مانند نور خورشید، لامپ، آتش یا نورهای سحرآمیز) از کجا به صحنه می‌تابند، چگونه با سطوح برخورد می‌کنند و بازتاب می‌شوند.

در بازی‌های بلادرنگ، به‌دلیل محدودیت‌های پردازشی، استفاده از نورپردازی کامل مبتنی بر فیزیک دشوار است. در عوض، از تکنیک‌هایی مانند نورپردازی پیش‌محاسبه‌شده (baked lighting)، نورهای نقطه‌ای (point lights)، نورهای جهت‌دار (directional lights) و سایه‌های نرم استفاده می‌شود. همچنین فناوری‌های پیشرفته‌تری مانند Ray Tracing برای شبیه‌سازی واقع‌گرایانه بازتاب نور در زمان واقعی در حال گسترش هستند که البته نیازمند سخت‌افزار قدرتمند و کارت‌های گرافیک نسل جدید مانند RTX انویدیا یا RDNA AMD است.

۳. تکسچرینگ (Texturing):

پس از مدل‌سازی، سطوح اشیاء نیاز به ظاهر واقع‌گرایانه دارند. در اینجا فرآیند تکسچرینگ وارد عمل می‌شود. تکسچرها تصاویری دوبعدی هستند که روی سطوح مدل سه‌بعدی نقشه‌برداری می‌شوند تا حس ماده‌گرایی (Materiality) مانند فلز، سنگ، چرم یا پوست ایجاد شود. این فرایند با استفاده از نقشه‌برداری UV انجام می‌شود؛ یعنی هر نقطه از سطح مدل به نقطه‌ای در تصویر دوبعدی تکسچر مرتبط می‌شود.

در تکسچرینگ پیشرفته، از چندین نوع نقشه (Map) برای افزایش کیفیت بصری استفاده می‌شود. از جمله آن‌ها می‌توان به Diffuse Map (برای رنگ پایه)، Normal Map (برای ایجاد برجستگی‌های مجازی)، Specular Map (برای شدت بازتاب)، و Roughness Map (برای میزان زبری سطح) اشاره کرد. استفاده‌ی درست از این نقشه‌ها باعث می‌شود سطحی صاف به نظر برسد که دارای ترک، خش یا برجستگی‌های طبیعی است، بدون اینکه بار محاسباتی بالایی ایجاد کند.

۴. شیدینگ (Shading):

شیدینگ فرآیندی است که مشخص می‌کند نور و رنگ چگونه بر سطح هر پیکسل تأثیر بگذارد. برخلاف نورپردازی که مکان تابش نور را تعیین می‌کند، شیدینگ مشخص می‌کند نور چگونه رفتار می‌کند. این فرآیند توسط شیدرها (Shaders) انجام می‌شود که کدهایی کوچک اما قدرتمند هستند و روی GPU اجرا می‌شوند.

شیدرها به چند دسته تقسیم می‌شوند: Vertex Shader که موقعیت رأس‌ها را پردازش می‌کند، Fragment Shader که رنگ هر پیکسل را محاسبه می‌کند، و Compute Shader که عملیات عمومی محاسباتی را انجام می‌دهد. الگوریتم‌هایی مانند Phong Shading، Blinn-Phong و PBR (Physically Based Rendering) برای شبیه‌سازی واقع‌گرایانه‌ی رفتار نور و مواد استفاده می‌شوند. شیدینگ با استفاده از داده‌های نور، زاویه دید دوربین، تکسچر و ویژگی‌های سطحی، رنگ نهایی را برای نمایش روی صفحه تعیین می‌کند.

۵. رندرینگ (Rendering):

رندرینگ آخرین مرحله در زنجیره پردازش گرافیک بلادرنگ است. در این مرحله تمام داده‌های هندسی، نوری، بافتی و شیدینگ جمع‌آوری شده و توسط GPU به تصویر نهایی روی صفحه نمایش تبدیل می‌شوند. این تصویر همان چیزی است که کاربر نهایی در یک فریم از بازی مشاهده می‌کند. در بازی‌های بلادرنگ، این فرآیند باید ده‌ها یا حتی صدها بار در ثانیه انجام شود (معمولاً ۳۰ تا ۶۰ فریم در ثانیه)، بدون هیچ وقفه‌ای.

رندرینگ می‌تواند از تکنیک‌های متعددی مانند Forward Rendering (رندر مستقیم)، Deferred Rendering (رندر تاخیری) یا Hybrid Rendering استفاده کند. برخی از بازی‌ها از سیستم‌های LOD (Level of Detail) برای کاهش پیچیدگی مدل‌های دوردست استفاده می‌کنند. همچنین فیلترهای پس‌پردازش (Post-Processing) مانند Bloom، Motion Blur و Depth of Field به منظور بهبود کیفیت بصری بعد از رندر اولیه اعمال می‌شوند. رندرینگ نیازمند هماهنگی دقیق بین داده‌ها، حافظه، و زمان‌بندی GPU است تا تصویری بی‌نقص در اختیار کاربر قرار گیرد.

تمام این مراحل باید ده‌ها یا صدها بار در هر ثانیه تکرار شوند (معمولاً ۳۰ تا ۱۴۴ بار در ثانیه)، که تنها با کمک GPUهای قدرتمند امکان‌پذیر است. همچنین تکنیک ها و فناوری های دیگری وجود دارند که منجر به تسهیل استفاده هر چه بهتر در تجربه بازی ها می‌شوند که در ادامه به معرفی برخی از آن‌ها می پردازیم

رهگیری پرتو (Ray Tracing)

تکنیکی برای شبیه‌سازی فیزیکی مسیر نور که موجب ایجاد سایه‌ها، بازتاب‌ها و انکسارهای بسیار واقع‌گرایانه می‌شود. در گذشته این تکنیک تنها در رندرینگ آفلاین کاربرد داشت (مثلاً در فیلم‌های سینمایی)، اما با ظهور هسته‌های RT Core در GPUهای جدید مانند سری RTX، اکنون می‌توان آن را به‌صورت بلادرنگ در بازی‌ها نیز استفاده کرد.

شیدینگ مبتنی بر فیزیک (PBR)

مدل جدیدی از شیدینگ که در آن مواد (Materialها) مانند فلز، چوب یا پارچه بر اساس خواص فیزیکی واقعی نور را بازتاب می‌دهند. این مدل باعث افزایش چشمگیر واقع‌گرایی صحنه‌ها می‌شود.

Anti-Aliasing

Anti-Aliasing (پاد-همپوشانی) یک تکنیک پردازش تصویر است که برای کاهش اثر "aliasing" یا پله‌پله‌شدگی در تصاویر دیجیتال استفاده می‌شود. این پدیده عمدتاً در لبه‌های مورب یا منحنی در تصاویر، جایی که پیکسل‌ها به صورت گسسته نمایش داده می‌شوند، مشهود است. به دلیل محدودیت رزولوشن نمایشگرها یا تصاویر دیجیتال، خطوط شیب‌دار به جای اینکه صاف و یکدست باشند، به صورت مجموعه‌ای از مربع‌های کوچک پله‌پله نمایش داده می‌شوند که باعث ایجاد ظاهر ناهموار و غیرطبیقی می‌شود. Anti-Aliasing با هوشمندانه ترکیب رنگ‌های پیکسل‌های مجاور در مرزها، این پله‌ها را محو می‌کند و باعث ایجاد یک انتقال رنگی نرم‌تر و ظاهری صاف‌تر می‌شود. این کار با میانگین‌گیری از رنگ‌های پیکسل‌های لبه و پیکسل‌های اطراف آن انجام می‌شود، به طوری که هر پیکسل در مرز به جای داشتن یک رنگ واحد، ترکیبی از رنگ‌های همسایه را به خود می‌گیرد.

روش‌های مختلفی برای انجام Anti-Aliasing وجود دارد که هر کدام دارای مزایا و معایب خاص خود هستند. به عنوان مثال، روش‌های مبتنی بر نمونه‌برداری (Sampling) مانند SuperSampling Anti-Aliasing (SSAA) با رندر کردن تصویر با رزولوشن بالاتر از حد مورد نیاز و سپس کوچک کردن آن به رزولوشن نهایی، به کیفیت بسیار بالایی دست می‌یابند، اما به منابع محاسباتی زیادی نیاز دارند. در مقابل، روش‌های مبتنی بر پس‌پردازش (Post-Processing) مانند Fast Approximate Anti-Aliasing (FXAA) یا Temporal Anti-Aliasing (TAA) که پس از رندر شدن تصویر اعمال می‌شوند، کارایی بیشتری دارند و کمتر به قدرت پردازشی نیاز دارند، اما ممکن است در برخی موارد به اندازه روش‌های مبتنی بر نمونه‌برداری دقیق نباشند و باعث کمی تاری در تصویر شوند. انتخاب روش Anti-Aliasing مناسب به کاربرد و منابع در دسترس بستگی دارد؛ در بازی‌های ویدیویی که نیاز به فریم‌ریت بالا است، معمولاً از روش‌های سریع‌تر استفاده می‌شود، در حالی که در رندرینگ سه‌بعدی یا طراحی گرافیک که کیفیت نهایی از اهمیت بالاتری برخوردار است، روش‌های دقیق‌تر ترجیح داده می‌شوند.

Tessellation و LOD

GPU ها به‌طور پویا می‌توانند چندضلعی‌های اشیاء را بر اساس فاصله از دوربین تقسیم یا ساده‌سازی کنند. این کار برای بهینه‌سازی عملکرد و کاهش بار پردازشی در صحنه‌های پیچیده انجام می‌شود.

موتورهای بازی مانند Unreal Engine، Unity، CryEngine و Frostbite همگی به‌شدت به GPU وابسته‌اند. این موتورها از طریق APIهایی مانند DirectX، Vulkan و OpenGL یا CUDA و RTX، با GPU ارتباط برقرار کرده و عملکرد گرافیکی را مدیریت می‌کنند.

برای مثال، موتور Unreal Engine 5 با فناوری‌هایی مانند Nanite (رندر هندسه‌ی بسیار پرجزئیات) و Lumen (سیستم نورپردازی پویا و واقع‌گرایانه)، نیازمند GPUهایی با توان پردازشی بسیار بالا است.

یکی از مهم‌ترین عواملی که تجربه‌ی کاربر را در بازی تعیین می‌کند، نرخ فریم (FPS) است. FPS بالا (۶۰ یا ۱۲۰ به بالا) باعث روان‌تر بودن بازی و واکنش سریع‌تر کاربر می‌شود. همچنین رزولوشن بالاتر (مانند K۴) نیز موجب افزایش کیفیت بصری می‌شود، اما بار پردازشی GPU را نیز افزایش می‌دهد.

برای حفظ FPS مناسب در رزولوشن بالا، GPU باید بتواند در هر فریم، میلیون‌ها پیکسل را با دقت بالا پردازش کند. در این زمینه، فناوری‌هایی مانند DLSS (افزایش وضوح تصویر با کمک هوش مصنوعی) توسط NVIDIA توسعه یافته‌اند که با استفاده از Tensor Coreها، تصویر را با کیفیت بالا و بار پردازشی پایین‌تر تولید می‌کنند.

با ظهور تکنولوژی‌های جدید مانند بازی‌های ابری (Cloud Gaming)، محتوای تعاملی مبتنی بر هوش مصنوعی، و جهان‌های مجازی بزرگ‌تر (مانند Metaverse)، نقش GPUها نه تنها در رندرینگ بلکه در محاسبات هوش مصنوعی و شبیه‌سازی‌های پیچیده نیز پررنگ‌تر خواهد شد. ترکیب GPU با AI در بازی‌ها می‌تواند منجر به شخصیت‌های باهوش‌تر، فیزیک واقع‌گرایانه‌تر و تجربیات فراگیرتر شود.

یادگیری عمیق و هوش مصنوعی

استفاده از پردازنده‌های گرافیکی در حوزه‌ی یادگیری عمیق (Deep Learning) و هوش مصنوعی (AI) یکی از تحولات بنیادینی است که پیشرفت‌های چشمگیر چند سال اخیر در این زمینه‌ها را ممکن ساخته است. GPUها با معماری مبتنی بر پردازش موازی، قادر به انجام هم‌زمان هزاران عملیات ریاضی هستند و همین ویژگی آن‌ها را برای اجرای الگوریتم‌های پیچیده‌ی یادگیری عمیق که نیازمند پردازش حجم انبوهی از داده و پارامتر هستند، به گزینه‌ای ایده‌آل تبدیل کرده است. در مقابل، CPUها با وجود توانایی بالا در پردازش سریالی و مدیریت سیستم، توانایی کمتری در انجام حجم زیاد محاسبات هم‌زمان دارند. همین تفاوت در ساختار معماری باعث شده تا در بسیاری از کاربردهای هوش مصنوعی، GPU به ابزار اصلی تبدیل شود.

در یادگیری عمیق، مهم‌ترین عملیات‌ها شامل ضرب ماتریس‌ها، جمع بردارها، محاسبه گرادیان‌ها و به‌روزرسانی وزن‌ها در طول فرآیند آموزش مدل‌های عصبی است. این عملیات‌ها که عموماً در ابعاد بسیار بالا

انجام می‌شوند، نیازمند توان محاسباتی فوق‌العاده بالایی هستند. برای مثال، در آموزش یک شبکه عصبی کانولوشنی (CNN) با میلیون‌ها پارامتر و هزاران نمونه داده، در هر epoch باید هزاران بار عملیات‌های ماتریسی و عددی سنگین انجام گیرد. GPUها با برخورداری از هزاران هسته‌ی پردازشی سبک‌وزن، این عملیات‌ها را به صورت موازی اجرا کرده و سرعت آموزش را به‌طور چشمگیری افزایش می‌دهند.

کتابخانه‌هایی مانند NVIDIA CUDA و cuDNN به توسعه‌دهندگان اجازه می‌دهند که به صورت سطح پایین یا از طریق رابط‌های سطح بالا (مثل TensorFlow، PyTorch یا Keras) از توان GPU بهره‌برداری کنند. این کتابخانه‌ها توابع پایه‌ای برای اجرای سریع عملیات‌هایی مانند ضرب ماتریس، فیلتر کانولوشن، فعال‌سازی، نرمال‌سازی، و عقب‌پراکندگی (Backpropagation) را روی GPU پیاده‌سازی کرده‌اند. مثلاً در یک شبکه‌ی عصبی با لایه‌های پیچیده، با استفاده از cuDNN می‌توان بهینه‌ترین پیاده‌سازی عملیات‌های پایه را با سرعت و بهره‌وری بالا روی GPU اجرا کرد، بدون اینکه برنامه‌نویس نیاز به پیاده‌سازی مجدد الگوریتم‌ها داشته باشد.

در کاربردهای عملی، GPUها نقش کلیدی در آموزش شبکه‌های بزرگ مانند مدل‌های Transformer و شبکه‌های GAN ایفا می‌کنند. این مدل‌ها گاه به آموزش روی داده‌هایی در مقیاس ترابایت نیاز دارند و تنها با استفاده از GPUهای قدرتمند (و در برخی موارد خوشه‌های چند GPU یا حتی TPUها) می‌توان آن‌ها را در مدت زمان معقول آموزش داد. برای مثال، آموزش یک مدل GPT مانند ChatGPT با میلیاردها پارامتر، با CPU به ماه‌ها زمان نیاز دارد در حالی که با چندین GPU قدرتمند، این زمان به چند روز یا حتی چند ساعت کاهش می‌یابد.

علاوه بر مرحله‌ی آموزش، GPU در مرحله‌ی استنتاج (Inference) نیز کاربرد دارد. هنگامی که مدل آموزش دیده آماده شده و باید پاسخ‌گویی به ورودی‌های جدید را انجام دهد، باز هم GPU با اجرای سریع لایه‌های مدل، زمان پاسخ‌گویی را کاهش می‌دهد. در سیستم‌های بلادرنگ مانند تشخیص چهره، ترجمه‌ی هم‌زمان زبان، تشخیص گفتار و رانندگی خودران، استفاده از GPU برای پردازش سریع ورودی‌های حسگر و تولید خروجی ضروری است.

در سال‌های اخیر، شرکت‌هایی مانند NVIDIA به طور خاص معماری‌های GPU را برای کاربردهای هوش مصنوعی بهینه‌سازی کرده‌اند. برای مثال، معماری‌های Volta، Turing و Ampere دارای «Tensor Cores» هستند که واحدهای پردازشی ویژه‌ای برای ضرب ماتریس‌های کوچک و عملیات یادگیری عمیق فراهم می‌کنند. این واحدها به‌صورت سخت‌افزاری برای اجرای سریع عملیات‌های ضروری در شبکه‌های عصبی طراحی شده‌اند و به‌ویژه در یادگیری ماشین با دقت پایین‌تر (مثل FP16 یا INT8) به طرز چشمگیری عملکرد را بهبود می‌بخشند.

افزون بر این، پلتفرم‌هایی مانند NVIDIA DGX و ابررایانه‌هایی مانند Summit، مخصوص پردازش‌های یادگیری عمیق با بهره‌گیری از ده‌ها یا صدها GPU طراحی شده‌اند. این زیرساخت‌ها با ایجاد امکان

موازی‌سازی توزیع‌شده (Distributed Training) روی مدل‌های عظیم، آموزش را در مقیاس‌های کلان ممکن می‌سازند. در واقع، ترکیب GPUهای متعدد با معماری‌هایی چون NVLink و تکنیک‌هایی مانند مدل‌سازی توزیعی (Model Parallelism) و داده‌محور (Data Parallelism)، افق‌های جدیدی برای توسعه مدل‌های هوش مصنوعی باز کرده است.

ظهور TPU و NPU و دیگر واحد های پردازشی

با رشد هوش مصنوعی و ظهور مدل‌های پیچیده شبکه‌های عصبی، نیاز به سخت‌افزارهایی که به صورت اختصاصی برای این نوع محاسبات بهینه‌سازی شده باشند، آشکار شد.

TPU (Tensor Processing Unit)

توسط گوگل و به طور خاص برای تسریع عملیات یادگیری ماشین (Machine Learning) و به خصوص چارچوب TensorFlow توسعه یافته است. TPUها برای انجام محاسبات ماتریسی و برداری که در شبکه‌های عصبی بسیار رایج هستند، بهینه‌سازی شده‌اند. معماری خاص TPU، به نام "آرایه سیستولی" (Systolic Array)، به آن اجازه می‌دهد تا هزاران عملیات ضرب و جمع ماتریسی را به صورت همزمان و بدون نیاز مکرر به دسترسی حافظه انجام دهد. این ویژگی باعث می‌شود TPUها در مقایسه با CPU و GPU، هم کارایی بسیار بالاتری داشته باشند و هم مصرف انرژی کمتری در وظایف یادگیری عمیق از خود نشان دهند. گوگل از TPU در محصولات و سرویس‌های ابری خود مانند Google Translate، Google Assistant و Google Photos به طور گسترده استفاده می‌کند.

NPU (Neural Processing Unit)

این واحدها به طور خاص برای اجرای عملیات مرتبط با هوش مصنوعی و یادگیری عمیق، به ویژه در دستگاه‌های کم‌مصرف مانند گوشی‌های هوشمند، تبلت‌ها، دستگاه‌های اینترنت اشیا (IoT) و خودروهای هوشمند طراحی شده‌اند. هدف اصلی NPU، پردازش محلی (On-device) عملیات هوش مصنوعی است، به این معنی که دستگاه می‌تواند وظایف هوش مصنوعی را بدون نیاز به اتصال دائم به اینترنت و پردازش ابری انجام دهد. NPUها با سرعت بخشیدن به وظایفی مانند تشخیص چهره، پردازش زبان طبیعی، بهبود کیفیت عکس و فیلم و مدیریت هوشمند باتری، تجربه کاربری را به طور چشمگیری ارتقا می‌دهند.
شرکت‌هایی مانند هواوی با تراشه Kirin 970 و اپل با تراشه‌های سری A Bionic، پیشگامان استفاده از NPU در دستگاه‌های موبایل بوده‌اند. سرعت پردازش NPU در مقایسه با هسته‌های سنتی CPU می‌تواند تا ۱۰ برابر بیشتر باشد و در عین حال مصرف انرژی بسیار کمتری دارد.

علاوه بر TPU و NPU، واحدهای پردازشی تخصصی دیگری نیز در حال توسعه هستند که هر کدام برای کاربردهای خاصی بهینه‌سازی شده‌اند

DPU (Data Processing Unit)

این واحدها بیشتر در مراکز داده و محیط‌های ابری به کار می‌روند و برای مدیریت و پردازش حجم عظیمی از داده‌ها و عملیات شبکه طراحی شده‌اند. DPUها می‌توانند بار پردازش شبکه و ذخیره‌سازی را از روی CPU اصلی بردارند و بهینگی و سرعت کلی سیستم را افزایش دهند.

QPU (Quantum Processing Unit)

در مقیاس بسیار متفاوتی، QPUها در حال حاضر در مراحل اولیه توسعه هستند و به جای بیت‌های سنتی، از "کیوبیت‌ها" برای انجام محاسبات استفاده می‌کنند. کامپیوترهای کوانتومی پتانسیل حل مسائل بسیار پیچیده‌ای را دارند که حتی قدرتمندترین سوپرکامپیوترهای امروزی نیز قادر به حل آن‌ها نیستند، از جمله توسعه مواد جدید، کشف داروها و بهینه‌سازی الگوریتم‌های هوش مصنوعی.

ظهور این واحدهای پردازشی تخصصی نشان‌دهنده یک تغییر پارادایم در صنعت محاسبات است. به جای تکیه صرف بر پردازنده‌های عمومی، ما به سمت معماری‌های ناهمگن (Heterogeneous Architectures) حرکت می‌کنیم که در آن CPU، GPU، TPU و NPU در کنار هم کار می‌کنند تا بهترین عملکرد را برای وظایف مختلف ارائه دهند. این همگرایی، نه تنها به افزایش کارایی و کاهش مصرف انرژی کمک می‌کند، بلکه راه را برای نسل جدیدی از برنامه‌های هوش مصنوعی هموار می‌سازد که می‌توانند به صورت محلی، سریع‌تر و با هوشمندی بیشتری عمل کنند. آینده محاسبات، بی‌شک در گرو همکاری هوشمندانه این مغزهای سیلیکونی خواهد بود.

استخراج رمزارز

در دنیای امروز، رمزارزها (Cryptocurrencies) به عنوان نوعی پول دیجیتال ظهور کرده‌اند که با استفاده از الگوریتم‌های رمزنگاری پیچیده، تراکنش‌های مالی را به شکلی ایمن، غیرمتمرکز و شفاف مدیریت می‌کنند. بیت‌کوین، اتریوم و لایت‌کوین تنها چند نمونه از شناخته‌شده‌ترین این ارزهای دیجیتال هستند که پایه‌های یک اقتصاد نوین را بنا نهاده‌اند. یکی از مفاهیم کلیدی در پس این شبکه‌ها، فرآیندی به نام استخراج (Mining) است.

در قلب شبکه‌های رمزارزی، استخراج فرایندی است که طی آن شرکت‌کنندگان با اختصاص قدرت پردازشی دستگاه‌های خود، معادلات پیچیده ریاضی را حل می‌کنند. هدف از این کار، اضافه کردن بلاک‌های جدید به زنجیره بلوکی (Blockchain) است. در ازای این خدمت، ماینرها پاداشی به‌شکل رمزارز دریافت می‌کنند. این پاداش نه تنها انگیزه‌ای برای مشارکت در شبکه است، بلکه مکانیسمی برای تولید رمزارزهای جدید نیز محسوب می‌شود.

امنیت و صحت شبکه‌های بلاک‌چین به الگوریتم‌هایی مانند اثبات کار (Proof of Work - PoW) وابسته است. در این الگوریتم، ماینرها در یک رقابت مداوم، مسئله‌ای ریاضی را حل می‌کنند که حل آن فقط از طریق آزمون و خطای بسیار زیاد امکان‌پذیر است. این مسئله اغلب شامل یافتن مقدار "نانس (nonce)" مناسب برای تولید هش خاصی از داده‌های بلاک است. برای مثال، استخراج یک بلاک بیت‌کوین می‌تواند نیازمند تریلیون‌ها عملیات هش SHA-256 باشد. در این حوزه، به پردازش یک تریلیون هش در یک ثانیه "تراهش (Terahash)" گفته می‌شود و کارت‌های گرافیک (GPU) بر اساس همین معیار در دنیای رمزارزها مقایسه می‌شوند.

معمولاً، استخراج‌کنندگان چندین GPU را در یک دستگاه به نام ریگ استخراج (Mining Rig) نصب کرده و به صورت مداوم آن را در شبکه‌ی بلاک‌چین یا استخرهای ماینینگ فعال نگه می‌دارند.

در سال‌های اخیر، با افزایش چشمگیر قیمت رمزارزها و رونق فعالیت‌های استخراج، بازار کارت گرافیک با یک کمبود جهانی بی‌سابقه مواجه شد. ماینرها کارت‌های گرافیک را در مقیاس انبوه خریداری می‌کردند، که منجر به افزایش شدید قیمت و دشواری دسترسی برای مصرف‌کنندگان سنتی این بازار مانند گیمرها، طراحان گرافیک و پژوهشگران هوش مصنوعی شد.

این وضعیت، شرکت‌های بزرگی مانند NVIDIA و AMD را بر آن داشت تا برای حفظ تعادل بازار و حمایت از مشتریان اصلی خود، اقداماتی را برای جلوگیری از مصرف ماینینگ‌محور GPUها انجام دهند. این اقدامات با واکنش‌های متفاوتی از سوی جامعه‌ی کاربران و ماینرها روبه‌رو شد.

در سال ۲۰۲۱، NVIDIA برای اولین بار در کارت‌های سری RTX 30 (به‌ویژه RTX 3060 و RTX 3070)، فناوری جدیدی به نام Lite Hash Rate (LHR) را معرفی کرد. این فناوری باعث می‌شد که در صورت شناسایی فعالیت ماینینگ، توان محاسباتی کارت به طور خودکار کاهش یابد—گاهی تا ۵۰٪ یا حتی کمتر. مکانیزم LHR ترکیبی از نرم‌افزار و سخت‌افزار بود: درایورهای رسمی هنگام شناسایی فعالیت‌های مربوط به استخراج رمزارز (مثلاً اجرای الگوریتم Ethash) نرخ هش را کاهش می‌دادند، و بخش‌هایی از BIOS کارت گرافیک نیز به صورت داخلی رفتار کارت را تنظیم می‌کردند تا در برابر دستکاری مقاومت کند.

البته، جامعه‌ی توسعه‌دهندگان شخص ثالث مانند NiceHash به سرعت وارد عمل شدند و موفق شدند قفل‌های LHR را تا حد زیادی باز کنند. در نسخه‌های جدید ماینرها، نرخ هش کارت‌های LHR تقریباً به سطح اصلی خود بازگشت.

نتیجه گیری

در طول چند دهه گذشته، پردازشگرهای گرافیکی (GPUها) از قطعات تخصصی برای رندرینگ تصاویر به موتورهای محاسباتی قدرتمند و همه‌کاره‌ای تبدیل شده‌اند که ستون فقرات بسیاری از نوآوری‌های تکنولوژیک مدرن را تشکیل می‌دهند. این تحول، که با پیشرفت‌های معماری و نرم‌افزاری همراه بوده، نه تنها صنعت بازی را دگرگون کرده، بلکه انقلابی در حوزه‌های علمی، هوش مصنوعی و محاسبات با کارایی بالا (HPC) به وجود آورده است.

آینده برنامه‌نویسی GPU به همان اندازه که درخشان است، چالش‌برانگیز نیز خواهد بود. یکی از روندهای اصلی، حرکت به سمت معماری‌های چیپلت (Chiplet) است. این رویکرد، که در آن یک GPU از چندین دای (die) کوچک‌تر به جای یک دای بزرگ تشکیل می‌شود، امکان مقیاس‌پذیری بی‌سابقه، بهبود عملکرد و کاهش هزینه‌های ساخت را فراهم می‌آورد. برنامه‌نویسی برای این پلتفرم‌های پیچیده‌تر نیازمند ابزارها و مدل‌های جدیدی است که بتوانند از ارتباطات بین‌چیپلتی بهینه استفاده کنند و بار کاری را به‌طور هوشمندانه بین دای‌های مختلف توزیع کنند.

روند مهم دیگر، ادغام ناهمگون (Heterogeneous Integration) است؛ جایی که CPUها، GPUها، و سایر شتاب‌دهنده‌ها (مانند FPGAها یا پردازنده‌های تخصصی AI) به‌طور فیزیکی نزدیک‌تر و با پهنای باند بالا به یکدیگر متصل می‌شوند. این همگرایی سخت‌افزاری نیاز به مدل‌های برنامه‌نویسی یکپارچه‌تر دارد که بتوانند به‌طور شفاف بین منابع مختلف محاسباتی جابه‌جا شوند و از نقاط قوت هر یک بهره ببرند. ظهور استانداردهایی مانند oneAPI اینتل و تلاش‌هایی برای سازگاری بیشتر بین اکوسیستم‌های مختلف (مانند HIP در AMD) نشان‌دهنده این مسیر است.

زبان‌های برنامه‌نویسی و فریم‌ورک‌های سطح بالاتر نیز نقش پررنگ‌تری ایفا خواهند کرد. با توجه به افزایش پیچیدگی سخت‌افزار، توسعه‌دهندگان به ابزارهایی نیاز دارند که بدون درگیر شدن با جزئیات سطح پایین معماری، بتوانند از قدرت موازی GPUها استفاده کنند. این شامل پیشرفت در کامپایلرها برای بهینه‌سازی خودکار کد، توسعه زبان‌های دامنه‌خاص (DSLs) برای کاربردهای خاص (مانند بیوانفورماتیک یا مدل‌سازی مالی)، و گسترش قابلیت‌های فریم‌ورک‌های یادگیری عمیق برای پشتیبانی از معماری‌های جدید و الگوریتم‌های نوآورانه خواهد بود.

در نهایت، همگامی با پیشرفت‌های سخت‌افزاری، به‌ویژه در زمینه هوش مصنوعی مولد (Generative AI) و محاسبات کوانتومی هیبریدی (Hybrid Quantum Computing)، نیازمند نوآوری‌های مداوم در مدل‌های برنامه‌نویسی GPU خواهد بود. این مدل‌ها باید بتوانند محاسبات عظیم و پیچیده مورد نیاز برای نسل بعدی هوش مصنوعی و شبیه‌سازی‌های کوانتومی را با کارایی و بهره‌وری بی‌سابقه انجام دهند. با این تحولات، GPUها و برنامه‌نویسی آن‌ها نه تنها در مرکز نوآوری‌های تکنولوژیک باقی خواهند ماند، بلکه به پیش‌ران اصلی انقلاب‌های آتی در علم، فناوری و جامعه تبدیل خواهند شد.

منابع:

- Google. Gemini AI Model.
- OpenAI. ChatGPT AI Model.
- XAI. Grok AI Model.

- NVIDIA Corporation. CUDA Toolkit Documentation & Developer Resources.

- Advanced Micro Devices (AMD). ROCm Documentation & Developer Resources.

- Khronos Group. OpenCL Specification & Registry.

- OpenACC Organization. OpenACC Standard.

- Microsoft Developer Network (MSDN). DirectX Graphics and Gaming.

- Sanders, J., & Kandrot, E. (2010). CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional

- Kirk, D. B., & Hwu, W. W. (2017). Programming Massively Parallel Processors: A Hands-on Approach (3rd ed.). Morgan Kaufmann.

- Pytorch.org. PyTorch Documentation

- Abadi, M., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

- Ryoo, S., et al. (2008). Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. ACM SIGPLAN Notices, 43(11), 73-82