Streaming Architectures and Technology Trends

نویسنده : John Owens (جان آونز)
دانشگاه California, Davis
مترجم : سیدمرتضی مستجاب الدعوه

تکنولوژی مدرن به سازندگان پردازنده های امروزی اجازه می دهد تا منابع محاسباتی عظیمی در آخرین چیپ های خود وارد کنند. رقابت برسر این است که این معماری ها از افزایش ظرفیت به طرف افزایش کارآیی حرکت کنند. دهه ی آخر توسعه ی پردازنده های گرافیکی نشان می دهد که طراحان GPU به شدت در این امر موفق بوده اند. دراین مقاله به تجزیه وتحلیل تکنولوژی و روند های طراحی GPU ها یی که امروزه ساخته می شوند می پردازیم و درمورد انتظاراتمان از آینده صحبت می کنیم.

1. Technology Trends

ما هم مثل کاربران کامپیوتر عادت داریم که نسل جدید سخت افزارهای کامپیوتر سریعتر از نسل گذشته و دارای توانایی های بیشتر باشد(والبته دارای قیمت کمتر). این سرعت چشم گیر توسعه فقط به وسیله ی توسعه ی مداوم در تکنولوژی های اساسی ممکن است و این اجازه را به ما می دهد تا قدرت پردازش بیشتری بر روی چیپ ها جایگزین شود. هرسال درسازمان مسیر جهانی حرکت تکنولوژی نیمه هادی ها (ITRS)پیش بینی می شود که درصنعت نیمه رسانا تعدادی از ویژگی های مطلوب قابل اندازه گیری مثل اندازه ی ترانزیستورها و تعداد ترانزیستورها درهر چیپ و مصرف کلی انرژی در سال جاری چگونه تعییر خواهند کرد. این برآمدها تاثیر عظیمی بر روی کمپانی های سازنده ی چیپ , وسایل ساخت چیپ و همین طور طراحان چیپ های نسل بعد می گذارد. دربخش بعدی روش هایی را که در آینده اجرا خواهند شد و نقشی که این روش ها درتوسعه ی پردازنده های گرافیکی آینده ایفا خواهند کرد را شرح می دهیم.

1.1 Core Technology Trends

پردازنده های امروزی از میلیون ها وسیله برای سوییچ کردن به نام ترانزیستور ساخته شده اند. همین طور که تکنولوژی پیشرفت می کند این ترانزیستورها و اتصالات میان آنها درسطح کوچکتری ساخته می شوند. درسال Gordon Moore, 1965 اشاره کرد که تعداد ترانزیستورهایی که به صورت مقرون به صرفه می تواند در یک قالب پردازنده ی مجزا قرار گیرد هرسال دوبرابر می شود(Moore 1965). Moore پیش بینی کرد که این افزایش در آینده نیز وجود خواهد داشت. به این پیش بینی که بارها از آن نام برده شد در اصطلاح قانون Moore یا همان Moore`s Law می گویند که امروزه در هرسال تقریبا 50 درصد جزییات بیشتری بر روی یک قالب مجزای پردازنده قرار می گیرد. دراین 40 سالی که از پیش بینی Moore می گذرد تعداد ترانزیستورها در قالب پردازنده ها از 50 تا درسال 1965 به صدها میلیون در سال 2005 تغییرکرده است و ما می توانیم انتظار داشته باشیم این مقدار رشد در دهه ی بعدی نیز ادامه یابد.

درنسل جدید چیپ ها علاوه براین که تعداد ترانزیستورها افزایش یافت اندازه ی ترانزیستورها نیز کوچکتر شد. چون اندازه ی این ترانزیستورها کوچکتر شده بود این ترانزیستورها سریعتر از ترانزیستورهای نسل قبل عمل می کردند و این موضوع اجازه می داد

تا در مجموع نیز سریعتر کار کنند. ثبت شده است که سرعت ترانزیستورها 15 درصد در هرسال افزایش می یابد (Dally and Poulton 1998). در پردازنده های مدرن, یک سیگنال سراسری به نام Clock باعث انجام محاسباتی که در یک پردازنده اتفاق می افتد در همان زمان می شود ودرنتیجه استفاده کننده از پردازنده تاثیر افزایش سرعت ترانزیستور را در افزایش Clock مشاهده می کند. در مجموع افزایش تعداد ترانزیستور و Clock Speed با هم ترکیب می شوند و به افزایش توانایی پردازنده منجر می شوند. توانایی این پردازنده ها هرسال 71 درصد زیادتر می شود.با وجود این افزایش توانایی سالانه, ما می توانیم انتظار داشته باشیم هرسال, میزان محاسبات درهر چیپ, 71 درصد نسبت به سال پیش بیشتر باشد.

حافظه ی نیمه رسانای کامپیوتر, که از روش تقزیبا متفاوتی نسبت به پردازنده ی Logic ساخته می شود , با این وجود باز هم از پیشرفت های یکسان در تکنولوژی ساخت بهره می برند. ITRS پیش بینی کرده توانایی محصولی به نام-Dynamic random access Memory یا همان حافظه ی DRAM هرسه سال دو برابر می شود. کارآیی DRAM از دوراه قابل اندازه گیری است. یکی از طریق پهنای باند یا همان BandWidth که تعداد Data هایی که در هر ثانیه جابه جا می شوند را اندازه می گیرد و دیگر از طریق تاخیر زمانی یا همانLatency که با اندازه گیری مدت زمان میان این که Data ی مورد نظر درخواست شده و پس فرستاده می شود محاسبه می شود. سرعت افزایش کارآیی DRAM به اندازه ی سرعت افزایش توانایی پردازنده نیست. براساس ITRS 2003 پهنای باند DRAM , هر سال 25 درصد افزایش می یابد و تاخیر زمانی DRAM سالانه 5 درصد بهبود می یابد.

1.2 نتایج پایانی

در کل, بیشتر زمینه هایی (Trend) که شرح دادیم فقط درباره ی نکات مثبت صحبت کردیم و گفتیم که نسل جدید از لحاظ تکنولوژی تولید, توانایی پردازش , پهنای باند رم و تاخیرزمانی بهبود می یابد . برای مثال, این افزایش توانایی که در هر سال صورت به سمتی حرکت می کند که هر die (قطعه نازک مستطیلی از یک قرص نیمه هادی سیلیکان که به هنگام ساخت مدارهای مجتمع بریده شده یا لایه لایه می گردد) مجزا جزییات و مجتمع های فراوانی وجود داشته باشد. پنجاه سال پیش طراحان شروع کردند به قرار دادن واحد محاسبات Floating-Point یا همان Floating-Point arithmetic Unit بر روی die پردازنده و این تنها چیزی بود که در die پردازنده قرار می گرفت. امروزه, همین بخش کمتر از یک میلیمتر مربع اشغال می کند و می توان صدها عدد از همین بخش در همان die پردازنده قرار داد.

اگرچه که مهمترین نتیجه از این روند های تکنولوژی , تفاوت میان آنهاست. وقتی که یک بخش قابل اندازه گیری با سرعتی متفاوت از بقیه تغییر می کند احتیاج به دوباره فکرکردن درباره ی فرض های حاکم بر پردازنده ها و طراحی سیستم ها به وجود می آید. ما می توانیم سه تا موضوع کلی را تعیین کنیم که این موضو عات بر پیشرفت معماری GPU ها در آینده کمک می کنندو آنها عبارتند از محاسبات در مقابل ارتباط (Compute Vs. Communicate), تاخیرزمانی در مقابل پهنای باند (Latency Vs. BandWidth)وتوان(Power).

1.3 Compute Vs. Communicate

وقتی که به طور همزمان Clock Speed و اندازه ی چیپ افزایش می یابد , مدت زمانی که یک سیگنال در تمام چیپ حرکت می-کند (که برحسبClock Cycle که زمان میان دو تیک از Computer Clock است) افزایش می یابد. برروی سریعترین پردازنده های امروزی , فرستادن یک سیگنال از یک طرف چیپ به طرف دیگر به طور معمول چند Clock Cycle طول می کشد و این زمان با عرضه ی پردازنده های نسل بعد افزایش می یابد. این رویداد زمانی که افزایش ارتباط زمانی با مقدار محاسبات مقایسه می شود قابل توصیف است. درنتیجه , در آینده ,طراحان مقدار محاسبات را در ترانزیستورهای چیپ ها افزایش می دهند تا احتیاج به ارتباط های گران قیمت از میان برود. تاثیراحتمالی دیگر , افزایش تعداد محاسبات انجام شده در یک Word (یک قسمت کوچک از حافظه) از پهنای باند حافظه است.به عنوان مثال , اجازه بدهید سه محصول برتر اخیر nVIDIA یعنیGeForce FX 5800 محصول GeForce Fx 5950,2002 محصول 2003و6800 GeForce Fx محصول 2004 را با هم مقایسه کنیم. حال این کارت ها را در

قالب نقطه ی اوج کارآیی برنامه ریزی Floating-Point در مقابل حداکثر پهنای باند سراسری به حافظه مقایسه می کنیم. Geforce Fx 5800 از 2 عملیات Floating-Point برای هر Word از پهنای باند سراسری پشتیبانی می کند این موضوع در حالی است که Geforce Fx 5950 از 2.66 عملیات پشتیبانی می کند و Geforce Fx 6800 از تقریبا 6 عملیات می تواند پشتیبانی کند.ما می-توانیم انتظار داشته باشیم این روند توسعه در نسل های بعدی پردازنده ها نیز ادامه یابد.شکل 1 داده های گذشته را که در آن تعداد عملیات های Floating-Point در هر ثانیه و Memory Bandwidth قابل استفاده برای چند سری از معماری های GPU را نشان می دهد.

1.4 Latency Vs. BandWidth

gap های بین مسیر حرکتBandWidth وLatency(تاخیرزمانی) از مهمترین گردانندگان معماری های نسل آینده اند. چون تاخیرزمانی آهسته تر از BandWidth پیشرفت می کند(Patterson 2004), طراحان باید روش هایی ابداع کنند تا این مقدارهای فراوان تاخیرزمانی را هموار کند. این امر به وسیله ی انجام کارهای مفید در زمان هایی که منتظر Data های بازگشتی ازعملیات هایی زمانبر هستیم انجام می گیرد.

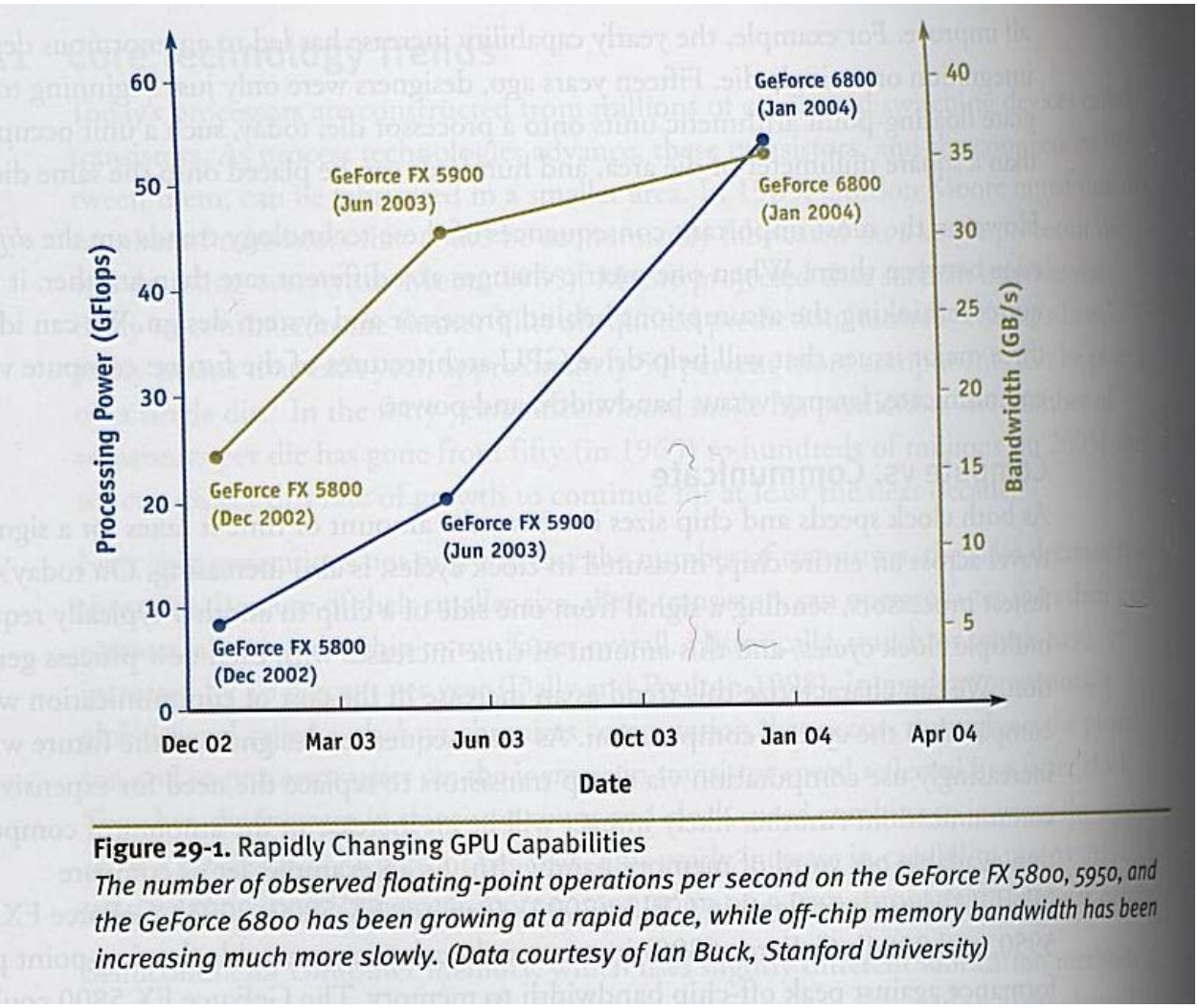

Figure 29-1. Rapidly Changing GPU Capabilities
The number of observed floating-point operations per second on the GeForce FX 5800, 5950, and the GeForce 6800 has been growing at a rapid pace, while off-chip memory bandwidth has been increasing much more slowly. (Data courtesy of Ian Buck, Stanford University)

1.5 توان

اگرچه ترانزیستورهای کوچک نسبت به ترانزیستورهای بزرگ تر توان کمتری دارند اما سرعت افزایش تعداد ترانزیستورها بر روی die یک پردازنده ی مجزا از سرعت مقدار توان کا هش یافته در هر ترانزیستور بیشتر است در نتیجه هر نسل جدید احتیاج به انرژی بیشتری دارد.ITRS بیشترین توان قابل قبول برای چیپ های سال 2004 به علاوه ی یکHeatSink را 158 وات تخمین زده است و این مقدار رفته رفته تا سال 2008 به 198 وات افزایش می یابد. این محدودیت توان یکی از مهمترین عوامل بازدارنده ی پردازنده ها در آینده خواهد بود. پس شایستگی در آینده ی نه چندان دور به وسیله ی تعداد عملیات هایی که در هر ثانیه انجام می شود معلوم نمی شود بلکه به جای این معیار از تعداد عملیات هایی که در هر ثانیه انجام می شود بر وات(توان مصرفی) استفاده می شود. شکل 2 تغییرات پیش بینی شده در توانایی , DRAM BandWidth,تاخیر زمانی DRAMو مقدار توان را در 10 سال آینده شرح می دهد.

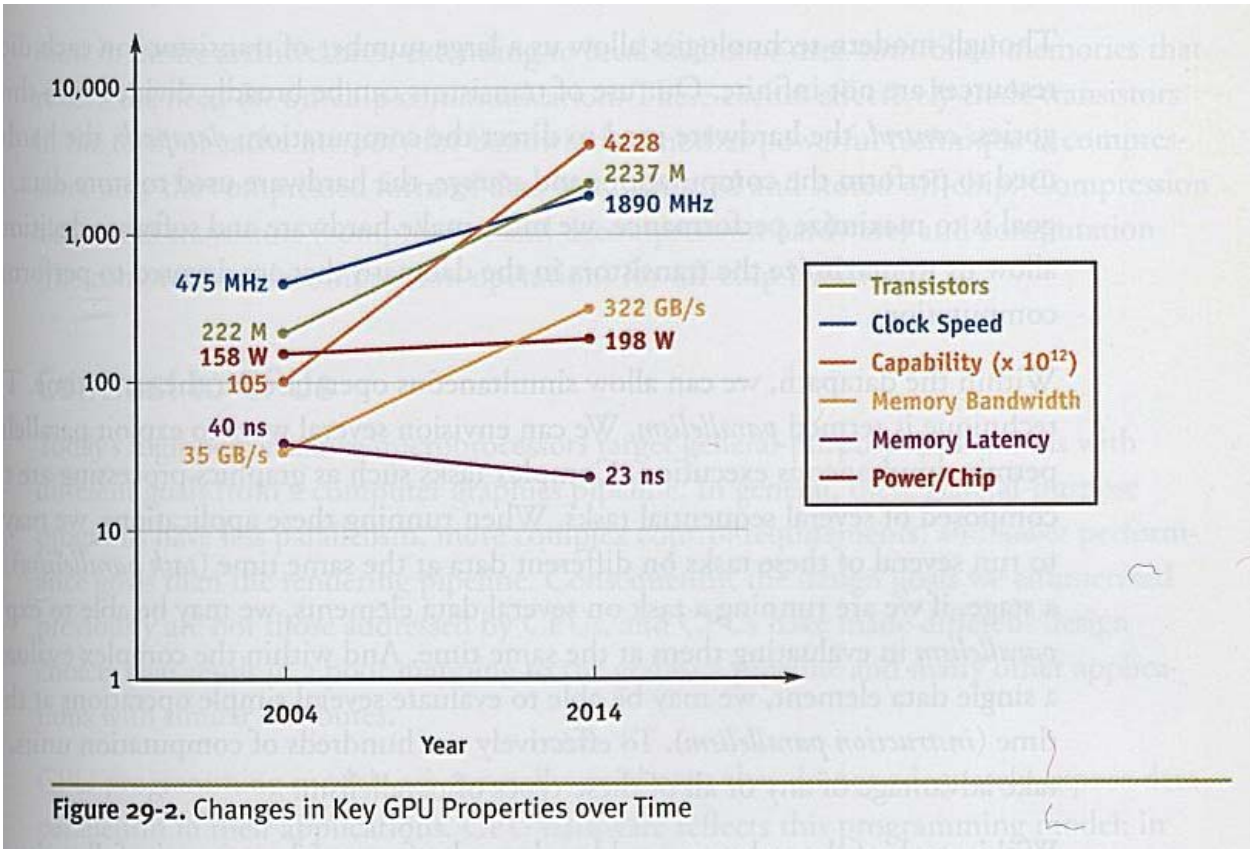

Figure 29-2. Changes in Key GPU Properties over Time

2. Keys to High Performance Computing

در بخش قبل, مشاهده کردیم که تکنولوژی هایمدرن اجازه می دهند هرنسل جدید از سخت افزارها توانایی بیشتری دارند. برای استفاده ی موثر از وسایل محاسباتی باید دو هدف را در نظر گرفت: اول باید وسایل محاسباتی را طوری سازمان دهی کنیم که بتوانیم حداکثر کارآیی رااز نرم افزار مورد نظر ببریم و دوم این که فراهم آوردن محاسبات فراوان واقعا صحیح نیست البته مدیریت صحیح اتصالات برای تغذیه وسایل محاسباتی بر روی چیپ بسیارمهم است. دراین قسمت تکنیک های محاسبه ی بهینه و اتصالات بهینه را مورد بررسی قرار می دهیم و سپس در این مورد این که چرا Microprocessor های مدرن یا همانCPU ها نمی توانند به خوبی ما را به این اهداف برسانند.

2.1 Methods for Efficient Computation

در بخش اول مشاهده کردیم که می توان صد ها بلکه هزاران بخش محاسباتی را بر روی یک die مجزا قرار داد. بهترین راه برای بهره بردن از ترانزیستورهای محاسباتی این است که از سخت افزارهایی که محاسبات را انجام می دهند حداکثر استفاده رابببریم و اجازه بدهیم چند بخش محاسباتی همزمان با هم, از راه موازی سازی (Parallelism) عمل کنند و مطمین شویم که این هر بخش محاسباتی با حداکثر بازده خود کار می کند.

اگرچه تکنولوژی مدرن به ما اجازه می دهد تا تعداد فراوانی ترانزیستور بر روی هر die قرارداد اما منابع ما نامحدود نیست. استفاده از ترانزیستورها به طور وسیع به سه طبقه ی تقسیم می شودControl سخت افزاری که برای جهت دهی به محاسبات استفاده م شود,DataPath, سخت افزاری برای بهینه سازی محاسبات و Storage, که سخت افزاری است برای ذخیره ی داده ها. اگر هدف ما به حداکثر رساندن کارآیی(Performance) است, ما باید سخت افزار و نرم افزار را در مسیری قرار دهیم که به اجازه می دهد که تعداد ترانزیستورهای موجود در DataPath که علاقه مند به انجام دادن محاسبات هستند را به حداکثر برسانیم.

ما می توانیم اجازه دهیم عملیات ها به طور همزمان در DataPath رخ دهند. این تکنیک در اصطلاح parallelism نام دارد. ما می توانیم راه های مختلفی را برای بهره بردن از parallelism تجسم کنیم و اجازه دهیم تا چند عمل به همزمان اجرا شوند. وظایف پیچیده مثلGraphic Processing (پردازش گرافیکی), که به دلیل ویژگی هایش ترکیبی از وظایف پشت سرهم و متوالی است. وقتی که این برنامه ها را اجرا می کنیم , ما می توانیم بعضی از این تکلیف ها را بر روی داده های مختلف در یک لحظه انجام دهیم (Task Parallelism). در یک مرحله, اگر در حال انجام یک وظیفه برای عناصر داده ای مختلف باشیم , ما می توانیم از parallelism برای محاسبه ی آنها به طور همزمان استفاده کنیم.ما می توانیم به جای انجام یک ارزیابی پیچیده از یک عنصر داده ای مجزا, چند عملیات ساده را به طور همزمان ارزیابی کنیم(Instruction Parallelism). برای اینکه صدها واحد محاسبه را به صورت موثر استفاده کنیم , می توانیم از برخی یا همه ی اشکال parallelism استفاده کنیم.

در هر ماموریت, می بایست بتوانیم تمام واحد های محاسباتی را به طور کامل برنامه ریزی کنیم در آن صورت است که می توان یک ماموریت را به خوبی انجام داد. البته ترانزیستورها می توانند راندمان بالاتری داشته باشند وا ین کار با تخصصی کردن آنها یا همان Specialization انجام می شود. اگر یک واحد محاسبه خاص, فقط یک شکل از اعمال محاسباتی را انجام دهد , می توان این واحد به آن شکل از اعمال محاسباتی خاص تخصیص یابد و در نتیجه به راندمان قابل توجهی دست یافت. برای مثال Triangle Rasterization, که عملیات است که فضای سه گوشی از صفحه ی نمایش را به قطعاتی کوچکی که این سه گوش را می پوشاند تبدیل می کند, زمانی که عملیات Rasterization برای سخت افزارهای تک منظوره صورت می گیرد مقدار بازده بسیار بیشتری را محقق می کند تا این که به جای آن سخت افزارهای قابل برنامه ریزی استفاده می شود.

2.2 Methods for Efficient Communication

همان طور که دربخش 1.2 مشاهد نمودید سرعت پیشرفت پهنای باند در خارج چیپ(off-Chip) کمتر از توانایی انجام عملیات های محاسباتی در داخل چیپ(On-Chip) است, پس باید پردازنده هایی که کارآیی فوق العاده دارند تا حد امکان این ارتباطات off-Chip را کاهش دهند. ساده ترین راه برای دست یابی به این امر, خلاص شدن از دست این ارتباطات می باشد. پردازنده های مدرن سعی می کنند تا می توانند داده های را بر روی چیپ نقل و انتقال کنند و ارتباطات خارج چیپ (Off-Chip) را فقط به آوردن و ذخیره ی داده های سراسری محدود می کنند.

راه دیگر برای کم کردن این ارتباطات فزاینده, از طریق caching می باشد یعنی یک کپی از جدیدترین داده ی استفاده شده از رم برروی چیپ ذخیره می شودو اگر این داده دوباره مورد نیاز بود دیگر لازم نیست که این داده از طریق ارتباطات خارج چیپ آورده شود. این شکل Caching داده ها بیشتر در معماری های آینده به کار می روند و به cache های محلی توسعه خواهند یافت که این Cache ها همان حافظه هایی هستند که توسط کاربر کنترل می شود و نیاز ارتباط های On-Chip را برطرف می کند.این Cache ها به صورت کاملا موثر , ترانزیستورها را به صورت حافظه ی Cache استفاده می کند . تکنیک قدرتمند دیگر Compression یا همان فشرده سازی است که فقط شکل فشرده شده ی Data جابه جا می شود و در خارج از چیپ ذخیره می شود.Compression هم توسط ترانزیستورها انجام می شود(سخت افزارهای Compression/Decompression لازم است) و محاسبات برای پهنای باند خارج از چیپ صورت می گیرد.(عملیات Compression/Decompression باید انجام شود).

2.3 Contrast to CPUs

ریزپردازنده های پرقدرت امروزی , چندین نوع برنامه را تحت پوشش قرار می دهند که هرکدام از آنها استفاده ی خاص خود را از Pipeline های گرافیکی کامپیوتر می کنند. درمجموع این برنامه های چندمنظوره Parallelism کمتری دارند, به کنترل پیچیده تری نیازمندند و کارآیی کمتری نسبت به Pipleline های Rendering برخوردارند. درنتیجه, اهداف طراحی که ما بر می شمردیم, آن اهدافی نیست که برای CPU مشخص شده بود وCPU ها طراحی های متفاوتی دارند که این طراحی ها, ترسیمات ضعیف در Pipelineهای گرافیکی و برنامه های دیگری با ویژگی های مشابه را نتیجه می دهد.

الگوهای برنامه نویسی CPU , عموما زنجیره ای(Serial) هستند که به قدر کافی موازی سازی داده ها(Data Parallelism) را در برنامه ها نمایش نمی دهند. خود سخت افزار CPU است که باعث به وجود آمدن این الگو می شود: معمولا, CPU در هر لحظه یک قسمت از داده ها را پردازش می کند و اصلا از موازی سازی داده ها(Data Parallelism) استفاده نمی کند. CPUها کارخوبی که انجام می دهند این است که از موازی سازی دستورات (Instruction Parallelism)استفاده می کنند و اخیرا در CPUها بخش هایی اضافه شده که علاوه بر دستورات, اجازه می دهند بعضی از داده ها نیز به صورت موازی اجرا شوند از جمله ی این بخش ها می توان به Intel’s SSEو PowerPC’s AltiVec اشاره کرد. البته باید گفت که این مقدار از موازی سازی(Parallelism) بسیار از موازی سازی (Parallelism)که در GPU صورت می گیرد کمتر است.

یک دلیل برای این که سخت افزارهای موازی(Parallel Hardwares) در CPU DataPath کمتر رایج هستند این است که طراح تصمیم گرفته است تا ترانزیستورهای بیشتری را برای کنترل سخت افزار صرف کند.برنامه های CPU نسبت به برنامه های GPU به کنترل های بسیار پیچیده تری نیاز دارند در نتیجه بخش عظیمی از ترانزیستورها و سیم ها وظیفه ی کنترل های بسیار پیچیده ای مثل تقسیم محاسبات(Branch Prediction) و اجرای خارج از نوبت(Out-Of-Order Execution) را بر عهده می گیرند در نتیجه بخش کمی ازDie مربوط به CPU صرف انجام محاسبات می شود.

چون CPU برنامه های چند منظوره را شامل می شود در نتیجه سخت افزار خاصی برای انجام کارخاصی را شامل نمی شود اما GPU می تواند کارخاصی راانجام دهد و در این راه از سخت افزارهایی که برای کارهای خاصی طراحی شده اند بهره ببردکه این موضوع باعث افزایش کارآیی GPU نسبت به حالتی که از یک سخت افزار همه منظوره قابل برنامه ریزی استفاده شود.

ودر آخر باید گفت که سیستم حافظه ای که برای CPU ها استفاده می شود برای بهره بردن از کمترین مقدار تاخیر زمانی طراحی شده است در حالی که سیستم حافظه ای که برای GPU استفاده می شود حداکثر ظرفیت را هدف می گیرد. عدم وجود موازی سازی(Parallelism) باعث شد تا برنامه های CPU مجبور شود تا ارجا به حافظه را هرچه سریعتر برگردانند تا CPU بتواند به کار خود را ادامه دهد. درنتیجه , سیستم حافظه ی CPU از لایه هایی ازحافظه ی cache (کسر عظیمی از ترانزیستورهای Chip تشکیل دهنده ی این بخش است) تشکیل شده است تا این تاخیر زمانی راکاهش دهد. اگرچه که Cache برای بسیاری از داده ها بی-فایده است(مثل ورودی های گرافیکی و آن دسته داده هایی که فقط یک بار در دسترس قرار می گیرند). در Pipeline های گرافیکی

, به حداکثر رساندن ظرفیت برای همه ی عناصر مهمتر از به حداقل رساندن تاخیرزمانی برای هر عنصر است و این موضوع باعث استفاده ی بهتر از سیستم حافظه ی GPU می شود و در نتیجه در مجموع کارآیی بیشتری را فرا هم می نماید.

3.Stream Computation

دربخش قبل مشاهده کردیم که ساختن پردازنده هایی با کارآیی فوق العاده امروزی به بهینه انجام دادن محاسبات و ارتباطات بهینه احتیاج دارد. بخشی از دلایل این که CPU ها را از رسیدن به بسیاری از کاربردهای پر بازده محروم می کند این است که شکل برنامه نویسی سریال یا زنجیره ای دارند(Serial Programming Model) و همین موضوع باعث می شود الگوهای موازی-سازی(Parallelism)و ارتباطات پوشیده باقی بمانند.در این بخش در مورد شکل برنامه نویسی جاری صحبت می کنیم (Stream Programming Model) که برنامه ها را به گونه ای سازماندهی می کند که برنامه اجازه یابد به حداکثر کارآیی در محاسبات و ارتباطات دست یابد. این شکل برنامه نویسی پایه ی برنامه نویسی پردازنده های گرافیکی(GPU)های امروزی است.

3.1 The Stream Programming Model

در شکل برنامه نویسی جاری (Stream) , همه ی داده هاجاری (Stream) درنظر گرفته می شود, یعنی به عنوان مجموعه ای ترتیبی از داده-هایی از یک نوع خاص شناخته می شوند. این نوع خاص می تواند ساده باشد(مثل Stream ای ازاعداد integer یا Floating-Point) یا پیچیده باشد(مثل Stream ای از نقاط(Points), سه گوش ها (Triangles) یا ماتریس های تبدیل). درحالی که یک Stream می تواند هر طولی داشته باشد, ما خواهیم دید که هر چه یک Streamطولانی تر باشد(صدها یا حتی تعداد بیشتری جز در یک Streamوجود داشته باشد) عملیات ها بر روی آنها از بازده بالاتری برخوردارند. عملیات هایی که بر روی Streamها اجازه داده می شوند عبارتند از Copy کردن, مشتق گرفتن زیر Stream (Deriving SubStream) از Stream ها , اندیس گذاری Stream ها بوسیله ی یک Stream ازindex ها و انجام دادن محاسبات بر روی Stream ها بوسیله ی Kernelها.

Kernel بر روی همه ی Stream ها عمل می کند و یک یا چند Streamبه عنوان ورودی دریافت می کند و یک یا چند Streamبه عنوان خروجی تولید می کند. ویژگی هایی که یک Kernel را به ما می شناسانند عبارت است از این که یک Kernel بر روی همه ی Streamهایی که شامل عناصرشان هشتندعمل می کند در نتیجه با عناصر مجزا مخالف است.عادی ترین استفاده ی یک Kernel این است که از آن برای انجام عملیات تابع بر روی هر کدام از عناصر موجود در یک Stream که به عنوان ورودی درنظرگرفته شده است استفاده شود.(یک عملیات نگاشت(MAP)) برای مثال یک Kernelکه برای انجام تبدیل به کار می رود ممکن است هر کدام از عناصر یک Stream از نقاط را به نقطه ای در دستگاه مختصات دیگر تصویر کند. Kernel های دیگری نیز وظیفه ی انبساط (Expansion)رابر عهده دارند(که بیش از یک عنصر خروجی برای هر عنصر ورودی تولید می کند), عده ای دیگر وظیفه ی کاهش(Reduction) یک Stream را برعهده دارند (که بیش از یک عنصر با هم ترکیب می شوند تا خروجی مجزایی را تولید کنند) و عده ای دیگر فیلتر(Filter) هستند(آنهایی که زیرمجوعه ای از ورودی ها را به عنوان خروجی تحویل می-دهند).

خروجی Kernel فقط عملیات صورت گرفته بر روی ورودی های Kernel است و درون یک یک Kernel محاسبات بر روی یک عنصر از Stream , هیچ وقت بر روی عنصر دیگر تاثیر نمی گذارد. این محدودیت ها دو امر مفید را به همراه دارند. یکی این که داده ای که برای اجرا شدن Kernelنیاز است در هنگام نوشته شدن و یا Compile کردن Kernel مشخص می شود. بنابراین یک Kernelمی تواند زمانی که عناصر ورودی و داده هایی که در خلال محاسبات به دست می آورد را به صورت محلی(Local) و یا ارجاع هایی عمومی کاملا کنترل شده(Carefully Controlled Global References) به صورت کاملا بهینه ذخیره کند. دوم این

که استقلال مورد نیازبرای انجام محاسبات بر روی عناصر مجزای Stream در درون یک Kernel مجزا به ما اجازه می دهد تا چیزی که به عنوان محاسبات سری Kernel (Serial Kernel Calculation)شناخته شده بر روی یک سخت افزار موازی صورت بگیرد.

در قالب برنامه نویسی جاری(Stream Programming) , نرم افزاری که مورد استفاده قرار می گیرد توسط متصل کردن چند Kernel به هم تولید می شود. به عنوان مثال, پیاده سازی Pipeline های گرافیکی در قالب Stream Programming باید Kernel برنامه ی Vertex , Kernel ی تولید Triangle , Clipping Kernel و خیلی Kernelهای دیگر را بنویسیم و سپس خروجی هر Kernel را به ورودیKernel بعدی متصل کنیم. شکل 3 چگونگی پیاده سازی Graphics Pipeline را در قالب Stream نشان می دهد. این قالب ارتباطات میان Kernel ها را به صورت آشکارا نمایش می دهد و از محل قرارگرفتن داده ها بین Kernel های اصلی در Graphics Pipeline استفاده می کند.

Graphics Pipeline به دلایل زیادی به خوبی بر قالب Stream منطبق است . همان طور که قبلا گفتیم Graphics Pipeline, از مراحل(Stages) محاسباتی که داده ها میان این مراحل در جریان اند ساخته شده است. این ساختمان مشابه مفهوم Stream و Kernel در قالب برنامه نویسی Stream می باشد. داده هایی که میان مراحل مختلف در Graphics Pipeline جاری می شوند, به شدت محلی شده اند به گونه ای که به محضی که داده ای در مرحله ای تولید می شود, در مرحله ی بعد مصرف می شود. درقالب برنامه نویسی به شکل Stream, Stream هایی که از میان Kernel ها می گذرند, رفتار یکسانی را نمایش می دهند.و محاسباتی که در هر مرحله از Pipeline صورت می گیرد برای انواع داده ای ابتدایی مختلف یکسان است واین موضوع اجازه می دهد تا این مراحل به راحتی به شکل Kernel ترسیم شوند.

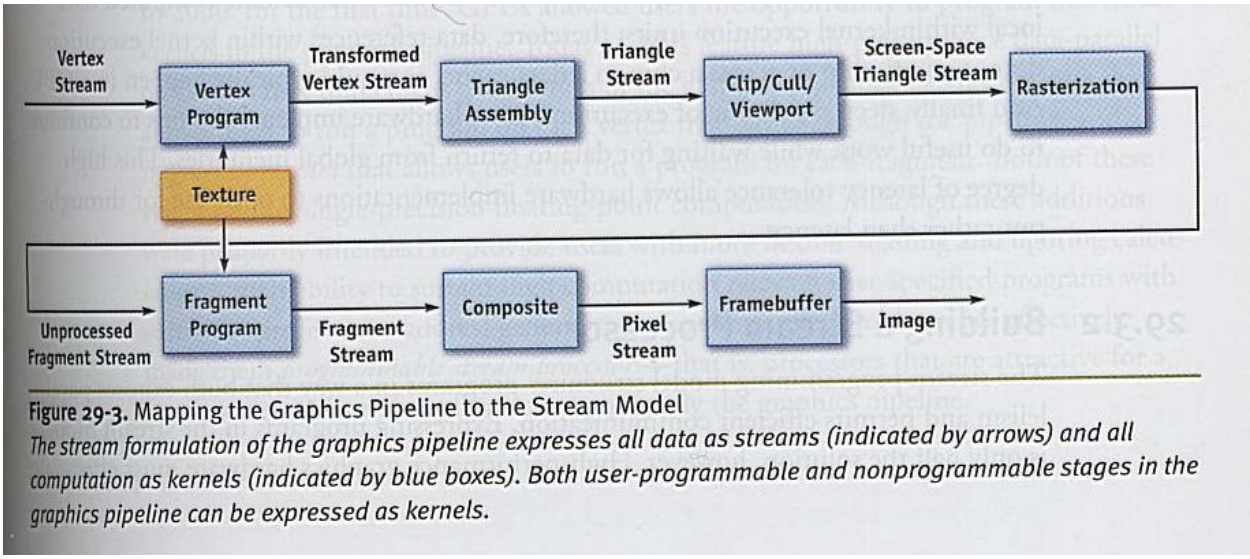

Figure 29-3. Mapping the Graphics Pipeline to the Stream Model

The stream formulation of the graphics pipeline expresses all data as streams (indicated by arrows) and all computation as kernels (indicated by blue boxes). Both user-programmable and nonprogrammable stages in the graphics pipeline can be expressed as kernels.

Efficient Computation

قالب Stream از چندین طریق ما را به محاسبات بهینه مجهز می کند. مهمترین عامل این بهینه سازی, این است که Stream ها باعث ظاهر شدن موازی سازی در کاربرد ها می باشد.چون Kernel ها بر روی تمام Stream عمل می کند و عناصر Stream می توانند به وسیله ی سخت افزارهای داده موازی(Data-Parallel Hardware) به صورت موازی پردازش شوند. Streamهای بلند با عناصرفراوان اجازه می دهند تا این مرحله از موازی سازی داده ها در حد زیادی بهینه شود.در هنگام پردازش یک داده ی مجزا, ما می توانیم از مرحله ی موازی سازی دستورات(Instruction-Level Parallelism) بهره ببریم. و چون در کاربردها از چندین Kernel استفاده می کنیم , چندین Kernel به شدت با هم Pipeline می شوند و به صورت موازی پردازش می شوند در نتیجه از موازی سازی وظایف(Task-Level Parallelism) استفاده می کنیم.

تقسیم کردن یک کاربرد مطلوب به Kernel ها به ما اجازه می دهد تا سخت افزارها را به گونه ای پیاده سازی کنیم تا به اجرای عملیات یک یا چند Kernel اختصاص یابند. سخت افزارها با کاربردخاص(Special-Purpose Hardware) , با کارآیی فوق-العاده اش در مقابل سخت افزارهای قابل برنامه ریزی, می توانند به خوبی در این قالب برنامه نویسی استفاده شوند.

سرانجام, فقط کنترل های ساده در هنگام اجرای Kernel اجازه ی جریان دارند و این موضوع این امکان را می دهد تا پیاده سازی سخت افزار به گونه ای باشد تا اکثر ترانزیستورهایش را به datapath Hardware اختصاص یابد نه به Control Hardware .

Efficient Communication

ارتباطات کارآمد یکی از اهداف اصلی قالب برنامه نویسی Stream است.اول, این که ارتباطات خارج از چیپ -off((Global) chip) زمانی که stream ها را به حافظه انتقال می دهد یا از آن خارج می کند بسیار بهینه تر عمل می کنند تا این که عناصر مستقل را به حافظه انتقال دهد یا از آن خارج کند.چون هزینه ی(این هزینه از زمان, پردازش یا هرچیز دیگر را شامل می شود) ثابتی برای آغاز کردن یک انتقال برروی Stream می تواند نادیده گرفته شود اما این هزینه بر روی هر عنصر مجزا وجود دارد. نکته ی دیگر این که سازماندهی کاربردها به صورت زنجیره ای از Kernelها اجازه می دهد تا نتایجی که در خلال اجرای Kernel ها به دست می آید بر روی چیپ(On-Chip) باقی بماند و به انتقال آنها به/از حافظه نیازی نداشته باشیم. Kernel های بهینه , سعی می کنند ورودی ها و داده های محاسبه شده ی میانی را درخلال واحد های اجرای Kernel به صورت Local(محلی) نگه دارند در نتیجه, ارجاع به داده ها(Data References) هیچ وقت به خارج یا Data Cache در کنار چیپ رخ نمی دهد در صورتی که این اتفاق در CPU رخ نمی دهد.در آخر باید گفت, Pipelining(انجام چند عمل در کنار هم برای کاهش زمان انجام عملیات) بسیار عمیق در هنگام اجرا, اجازه می دهد تا سخت افزار به گونه ای ساخته شوند تا زمانی که منتظر بازگشت Data از حافظه های سراسری هستند, سخت افزار به انجام کارهای مفید ادامه دهد.

3.2 Building a Stream Processor

قالب برنامه نویسی Stream , برنامه ها را به گونه ای سازماندهی می کند که هم موازی سازی(Parallelism) را گسترش دهد و هم ارتباطات بهینه را فراهم آورد. برنامه نویسی در قالب Stream تنها نیمی از راه حل می باشد.سخت افزارهای گرافیکی با کارآیی بالا می بایست به صورت کاملا بهینه هم از قدرت محاسباتی با کارآیی فراوان بهره ببرندو هم عملیات های محاسباتی راکاملا کارآمد انجام دهند که این دو موضوع به وسیله ی قالب Stream قابل بهره برداری می باشند. چگونه می توان ساختمان یک GPU(پزدارنده ی گرافیکی) را پیاده سازی کنیم تا مطمین شویم به بالاترین عملکرد ممکن دست یافته ایم؟

اولین قدم برای ساخت یک GPU با کارآیی فوق العاده, طرح ریزی Kernel ها مورد استفاده در Pipelineهای گرافیکی به واحدهای مستقل قابل استفاده بر روی یک چیپ مجزاست. درنتیجه هر Kernel روی بخش مجزایی از یک چیپ پیاده سازی شده است , این نوع سازماندهی را Task Parallel یا موازی کردن وظایف گویند که نه تنها باعث موازی سازی در سطح وظایف می شود(چون همه ی Kernel ها می توانند به صورت مجزا اجرا شوند) بلکه می توان سخت افزارهایی برای بخش های کاربردی تا Kernelخاص تخصیص داد. ساختار موازی سازی وظایف (Task Parallel Organization) باعث ایجاد ارتباطات کاملا موثر میان Kernel ها می شود و چون واحدهای کاربردی که از Kernel هایی در Graphic Pipeline خود استفاده می کنند که بر روی چیپ همسایه اند در نتیجه این Kernel ها می توانند بدون نیازبه دسترسی به حافظه Global به صورت کاملا مفید با یکدیگر ارتباط داشته باشند.

در هر مرحله از Graphic Pipeline که توسط واحد پردازشی روی چیپ انجام می شود, GPU استقلال خود را از هرکدام از عناصر Stream با پردازش چندین عنصر داده ای به صورت موازی آشکار می کند. ترکیب موازی سازی در سطح وظایف-(Task Level Parallelism) و در سطح داده ها(Data-Level Parallelism) باعث می شود تا GPU ها به صورت کاملا مفید از دوازده واحد کاربردی به صورت همزمان استفاده کنند.

ورودی به Graphics Pipeline باید توسط Kernel ها به صورت پشت سر هم انجام گیرد درنتیجه ممکن است یک عنصر هزاران چرخه را برای کامل شدن عملیات پردازش پشت سر گذارد. اگر حافظه ی مرجع با تاخیر زمانی بالا در عملیات پردازش عناصر معینی نیاز است, واحد پردازشی می تواند در خلال این که داده ها آورده (Fetch) می شوند به سادگی بر روی بقیه عناصر کار کند. این Pipeline های عمیق پردازنده های گرافیکی مدرن امروزی اند که عملیات های دارای تاخیر زمانی را به صورت کاملا موثر هموار می کنند.

سال ها بود که Kernel هایی که Graphics Pipeline را تشکیل می دهند در سخت افزار گرافیکی به صورت واحد های کاربردی ثابت پیاده سازی شدهه بودند و قابلیت برنامه نویسی توسط کاربر بسیار کم و یا اصلا نبود. در سال 2000, برای اولین بار GPUها به کاربران این فرصت را دادند تا Kernelهای منحصر به فردی بنویسند و در Graphics Pipeline از آن استفاده کنند.GPU های امروزی که از پردازنده هایی با قابلیت موازی سازی داده ها با کارآیی فوق العاده بهره می برند, از دو Kernel در Graphicss Pipeline خود استفاده می کنند: یکVertex Program که به کاربران اجازه می دهد تا یک برنامه را بر روی تک تک Vertex هایی که از Pipeline می گذرند, اجرا کند.Fragment Program که اجزه می دهد تا برنامه بر روی هر Fragment اجرا شود. هر دوی این مراحل اجازه ی انجام محاسبات اعشاری با دقت یکسان را دارند. اگر چه این موارد در ابتدا به قصد افزایش قابلیت سایه زنی انعطاف پذیر و برآورد نور پردازی ها تامین شدند اما توانایی آنها در رسیدن به سرعت های محاسباتی زیاد در برنامه هایی که توسط کاربرتعیین شده است همراه با دقت کافی برای آدرس دهی در مسایلی که به محاسبات همه منظوره (General-Purpose Computation) نیاز است باعث شد تا این پردازنده ها به صورت کاملا موثر به پردازنده های Stream قابل برنامه ریزی تبدیل شوند(Programmable Stream Processors). که پردازنده هایی اند که نسبت به Graphic Pipelineهای ساده, کاربردهای بسیار وسیع تری را دارند.

4.The Future and Challenges

مهاجرت GPUها به پردازنده هایStream ای که قابلیت برنامه نویسی دارند, نقطه ی اوجی در میان روش هایی که در دوران گذشته طی شده است را نشان می دهد. اولین روند, توانایی متمرکز کردن مقدار فراوانی محاسبات بر روی یک Die پردازنده ی مجزاست البته در کنار این موضوع باید به توانایی هاو استعدادهای طراحان GPU ها اشاره کرد که توانسته اند از این منبع محاسباتی به خوبی استفاده کنند. علم اقتصاد مقیاس که با تولید ده ها میلیون پردازنده پردازنده در هرسال پیوند خورده است باعث می شود تاقیمت یک پردازنده گرافیکی به اندازه کافی نزول کند تا آنجا که پردازنده ی گرافیکی, امروزه به عنوان یک بخش استاندارد و عادی کامپیوتر های Desktop درآمده است و افزایش معقولانه ی قدرت برنامه نویسی با دقت برای Pipeline باعث شده است تا گذار از یک پردازنده ی Hard-Wired(وسیله ی الکترونیکی که فقط حاصل از مدارهای الکترونیکی متصل به هم است و نرم افزار اصلا در آن تاثیری ندارد) ودارای استفاده ی خاص(Special-Purpose) به یک پردازنده ی بسیار قدرتمند و قابل برنامه ریزی که می توان در موارد بسیار وسیع از آن استفاده کرد, کامل شود .

اما , آیا اکنون ,می توان انتظار پیشرفت در پردازنده های گرافیکی آینده را داشت؟

4.1 Challenge: Technology Trends

هر نسل جدیدی از سخت افزارها در میان سازندگلن GPU رقابت جدیدی راه می اندازد و آنها در پی این موضوع اند که چگونه به صورت بهینه از منابع سخت افزاری جدید استفاده کرد تا در کارآیی و تعداد عملیات هایی که می توان انجام داد تحولی در جهت پیشرفت حاصل شود؟ ترانزیستورهای جدید صرف افزایش کارآیی در بخش بزرگی از کار می شوداین افزایش کارآیی از افزایش مقدار موازی سازی گرفته تا اضافه کردن عملیات جدید در Pipeline تاثیر خود را می گذارد. همچنین با تغییر تکنولوژی ها شاهد پیشرفت معماری ها نیز هستیم.

همان طور که در بخش 1.2 اشاره شد, معماری ها در آینده ,به صورت کاملا فزاینده از ترانزیستورها استفاده می کنند و این عمل را جایگزین نیاز به ارتباطات(Communication) می کنند.ما می توانیم انتظار تکنیک های Caching بسیار موثرتر را داشته باشیم که نه تنها ارتباطات خارج Chip (OFF-Chip Communication) را کاهش می دهند بلکه ارتباطات روی Chip (ON-Chip

(Communication) را نیز کم می کنند.ما شاهد این موضوع خواهیم بود که محاسبات(Computation) جای ارتباطات(Communication) را خواهند گرفت.برای مثال, استفاده از حافظه ی مربوط به Texture ها به عنوان یک جدول که دایم به آن مراجعه می شود از میان خواهد رفت و با ,محاسبه ی کاملا پویای(Dynamic) مقدارهای موجود در جدول هایی که به آنها مراجعه می شد, جایگزین خواهد شد. و همچنین به جای فرستادن Data به فاصله ی دوری بر روی Chip و به یک منبع محاسباتی وسپس دریافت حاصل, به سادگی می توانیم یک منبع دیگری بسازیم و محصولات را به صورت کاملا محلی(Locally) محاسبه کنیم. دراین دادوستد(Trade-Off) بین ارتباطات(Communication)و محاسبه مجدد/Cache (Recompute/Cache) , بیشتر دومی را انتخاب می کنیم.

افزایش قیمت ارتباطات(Communication) , بر روی معماری ریز پردازنده ها نیز تاثیر خواهند گذاشت, هم اکنون طراحان باید به صورت به روشنی باری زمان لازم برای ارسال داده ها درداخل یک Chip برنامه ریزی کنند; حتی زمان های مربوط به ارتباطات محلی, در محاسبات مربوط به زمان بندی ها قابل اهمیت می شوند.

4.2 Challenge:Power Management

ایده هایی که برای چگونگی بهره برداری از ترانزیستورهای GPU های نسل آینده, باید توسط واقعیتی به نام بهای این ترنزیستورها, تعدیل شود.مدیریت توان(Power Management) به بخش حیاتی از طراحی پردازنده های گرافیکی تبدیل شده است و هر نسل جدید از سخت افزار ها که عرضه می شوند, به توان بیشتری نیاز دارند. درآینده, ممکن است شاهد حرکت بزرگی به سمت مدیریت توان به صورت پویا (Dynamic Power Management) در حین عملیات های منحصر به فرد باشیم . مثل افزایش میزان طراحی های منحصر به فرد یا نگهدارنده ی توان(Power-Aware) برای قسمت های پرمصرف(Power-Hungry) پردازنده ی گرافیکی.و سیستم مدیریت خنک کننده ی بسیار پیشرفته تر برای پردازنده های گرافیکی پیشرفته تر.روند تکنولوژی نشان می دهد که احتیاج به توان بیشتر ادامه خواهد یافت و تا رسیدن به نسل آینده ی چیپ ها افزایش خواهد یافت پس کار مداوم در این بخش, موضوع مهمی باقی خواهد ماند.

4.3 Challenge : Supporting More Programmability and Functionality

در حالی که نسل کنونی سخت افزارهای گرافیکی, اساسا قابل برنامه ریزی بیشتری نسبت به نسل گذشته دارند, اما شکل اصلی قابلیت برنامه ریزی این GPU ها از شکل ایده آل خود بسیار دور است.یک قدم به سمت حرکت در جهت این روش(Trend) افزایش کارآیی و انعطاف پذیری در دو واحد کنونی قابل برنامه ریزی است (Vertex , Fragment). احتمالا در آینده شاهد مجموعه دستور العملها در کنار یکدیگر جمع شوند و کارآیی را افزایش دهند و ظرفیت روند کنترل آنها خیلی کلی تر خواهد شد. حتی ممکن است شاهد سخت افزارهای قابل برنامه ریزیی باشیم که در انجام یک عمل بین این دو مرحله مشترک اند تا این که بتوان هر چه بهتر از این منابع استفاده کرد. معماری GPU خیلی متفکرانه خواهد بود اگرچه که این پیشرفت ها, در کارآییGPU برای وظایفی که به هسته محول می کنند, بی تاثیر است. انتخاب دیگگر گسترش دادن قابلیت برناه نویسی برای واحدهای مختلف است. چون اشکال هندسی اولیه به صورت خاص با کمک گرفتن از قابلیت برنامه نویسی ساخته می شوند, احتمالا در آینده نزدیک شاهد پردازش های قابل برنامه ریزی بر روی pixel, triangle, surface خواهیم بود.

در عین حال که شرکت های فروشنده ی GPU , در حال پشتیبانی ازتعداد Pipeline های کلی(General Pipelines) بیشتری هستند و محاسبات shader پیچیده تر و گوناگونی استفاده می کنند, بسیاری از محققان کارهایی خارج از محدوده ی فعالیت هایPipeline های گرافیکی (Graphics Pipeline) محول می کنند.محاسبات همه مظوره(General Purpouse Computation) بر روی جامعه ی GPU ها (GPGPU) , توانست تا مسایلی موجود در زمینه هایی مانند شبیه سازی بصری(Visual Simulation) , پردازش تصاویر (Image Processing) , روش های عددی و پایگاه های داده ای را با موفقیت بر روی سخت افزارهای گرافیکی حل کند. می توانیم انتظار داشته باشیم تا این تلاش ها با ادامه ی افزایش کارآیی و قدرت GPU ها افزایش یابد.

در گذشته شاهد وظایفی در GPU ها بودیم که از وظایفی که قبلا مربوط به CPU ها بوده است, نتیجه شده اند. مشتریان قدیمی سخت افزار های گرافیکی نمی توانستند پردازش های هندسی خود را بر روی پردازنده های گرافیکی انجام دهند و همین 5 سال پیش بود که توانستند کل Pipeline های گرافیکی را روی یک چیپ مجزا سوار کنند.اگرچه , از آن به بعد, هدف اصلی جهت افزایش وظایف GPU ها, حزکت به سمت قابلیت برنامه نویسی برای Pipeline های گرافیکی شد , اما نباید انتظار داشته باشم, که فروشندگانGPU , تلاش های خود را برای مشخص کردن وظایف بیشتر برای تکمیل GPU ها متوقف کنند. به ویژه که در دنیای بازی های امروزی, محاسبات فراوانی لازم است که برای physics و هوش مصنوعی صورت پذیرد. شاید این محاسبات جذب کننده ی پزدازنده های نسل آینده باشند.

4.4 Challenge: GPU Functionality Subsumed by CPU(or Vice Versa)

ما می توانیم مطمین باشیم که سازندگان CPU ,درحالی کهGPU ها از قدرت محاسباتی بیشتر و توانایی های بیشتر در Chip های آینده خود استفاده میکنند ,متوقف نخواهند شد. تعداد ترانزیستورها همیشه در نسل های پردازشی در حال افزایش اند واین موضوع ممکن است که باعث کشمکش و مبارزه بین سازنده های CPU و GPU شود.آیا در آینده هسته ی کامپیوترها , CPU است؟ وسرانجام GPU یا عملیات های Stream با CPU به هم می پیوندند؟ یا این که نه! قلب کامپیوتر های آینده یک GPU خواهد بود که با ویژگی های CPU پیوند خورده است؟پرسش هایی موثر مانند این پرسش هاست که نسل بعدی معماری های پردازنده ها را به حرکت وا می دارد. وقتی به جلو نگاه می کنیم, یک آینده هیجان انگیز را می بینیم!

نظر مترجم : همان طور که می بینید در گذشته سوالی مطرح بوده که GPU یا CPU است که درآینده هسته ی کامپیوتر ها خواهد شد, و امروز شاهد این موضوع هستیم که AMD شرکت ATI را خریداری می کند و Intel با nVIDIA متحد می شود و طبق برنامه ی اعلام شده پردازنده هایی به زودی به یازار عرضه خواهند شد که نقش GPUرا نیز دارند و پیش بینی نویسنده, حقیقت پیوست.

5.References

Dally, William J., and John W. Poulton. 1998. Digital Systems Engineering. Cambridge University Press

ITRS. 2003. International Technology Roadmap for Semiconductors **http://public.itrs.net**

Kapasi, Ujval J., Scott Rixner, William J. Dally, Brucek Khailany, Jung Ho Ahn, Peter Mattson, and John D. Owens. 2003. “Programmable Stream Processors.” IEEE Computer, pp. 54-62

Moore, Gordon. 1965. “Cramming More Components onto Integerated Circuits.” Electronics 38(8). More Information available at **http://www.intel.com/research/silicon/mooreslaw.htm**

Owens, John D. 2002. “Computer Graphics on a Stream Architecture.” Ph.D. Thesis, Stanford University, November 2002.

Patterson, David A. 2004. “Latency Lags Bandwidth.” Communications of the ACM 47(10), pp. 71-75.