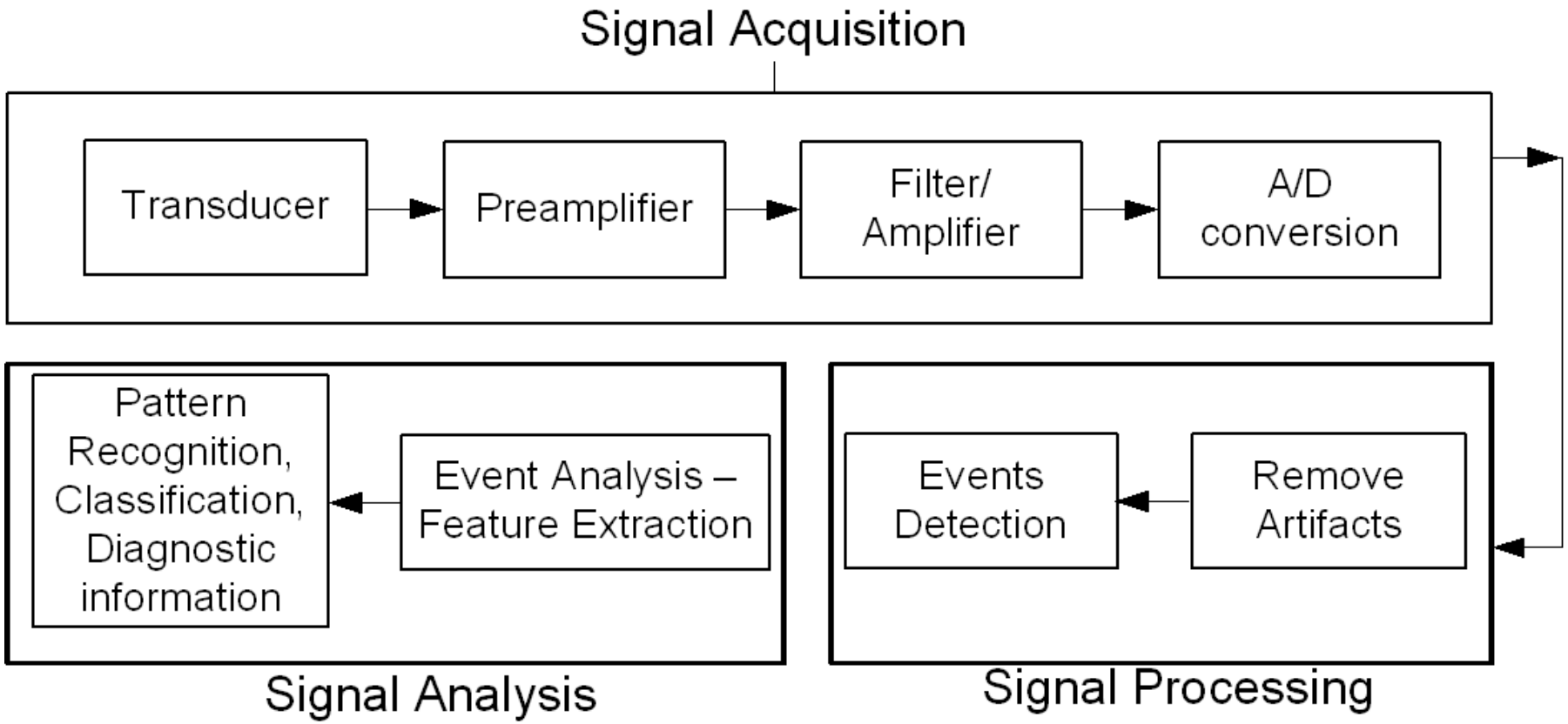
多媒体通信
Multimedia Communications

多媒体计算→异构计算

2015年10月26日

信号处理数据分析的统一模型

信号获取→数字信号处理→信息分析



这样的系统需要什么样的计算能力

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

多媒体计算需要什么？

- 多媒体数据处理的特点
  - 数据量大，但可并行化
  - 计算密集
  - 浮点数
  - 超越函数
- 提高多媒体数据处理的可能途径
  - 提高CPU计算能力
    - 多个运算单元（空间换时间）→多核
    - 工作时钟→主频
    - GPU
  - 提高数据进出CPU的速度
    - CPU←→Memory
    - CPU←→外设

多媒体数据处理对计算能力的要求

- 整数运算能力
  - 例：直方图均衡

- 浮点数运算能力
  - 例：颜色空间变换

$$\begin{cases} Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\ U &= -0.147 \times R - 0.289 \times G + 0.436 \times B \\ V &= 0.615 \times R - 0.515 \times G - 0.100 \times B \end{cases}$$

- 超越函数运算能力
  - 例：坐标旋转
    - x1=cosα*x-sinα *y;
    - y1=cosα*y+sinα*x;
  - 例： FFT
  - 例： DCT

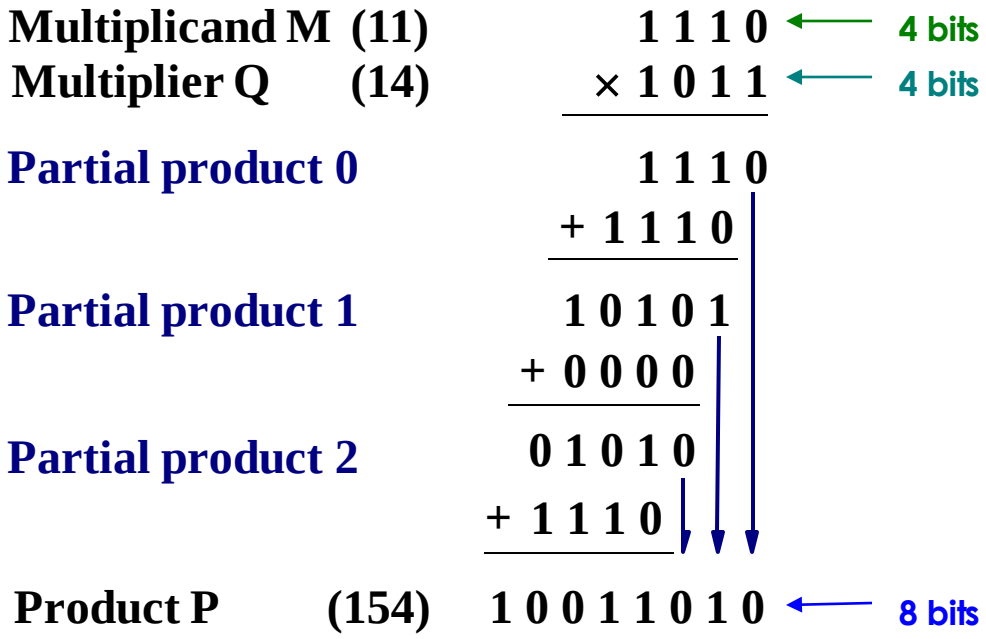
整数/浮点数运算能力
MIPS、FLOPS

MIPS表征了整数加法的运算能力

- MIPS: Millions of instructions per second
  - 1978年Intel 8086，0.33 MIPS@5MHz
  - 1989年Intel i486DX，8.7 MIPS@25MHz
  - 2015年Intel i7 6700k，24182 MIPS@4GHz
- FLOPS: FLOating-point operations per second
  - 1989年Intel i486DX，50000 FLOPS
  - 2015年Intel i7 6700k(HD Graphics 530，441.6G FLOPS)
  - iPhone6 (PowerVR GX6650，115.2GFLOPS@300MHz)

手机	价格￥	GPU	FLOPS
小米4	1500	Adreno 330	129.6G
魅蓝2	800	Mali T720	81.6G
P8	3500	Mali-T628	76.8G

整数/浮点数运算能力
乘法的计算成本（乘法原理）

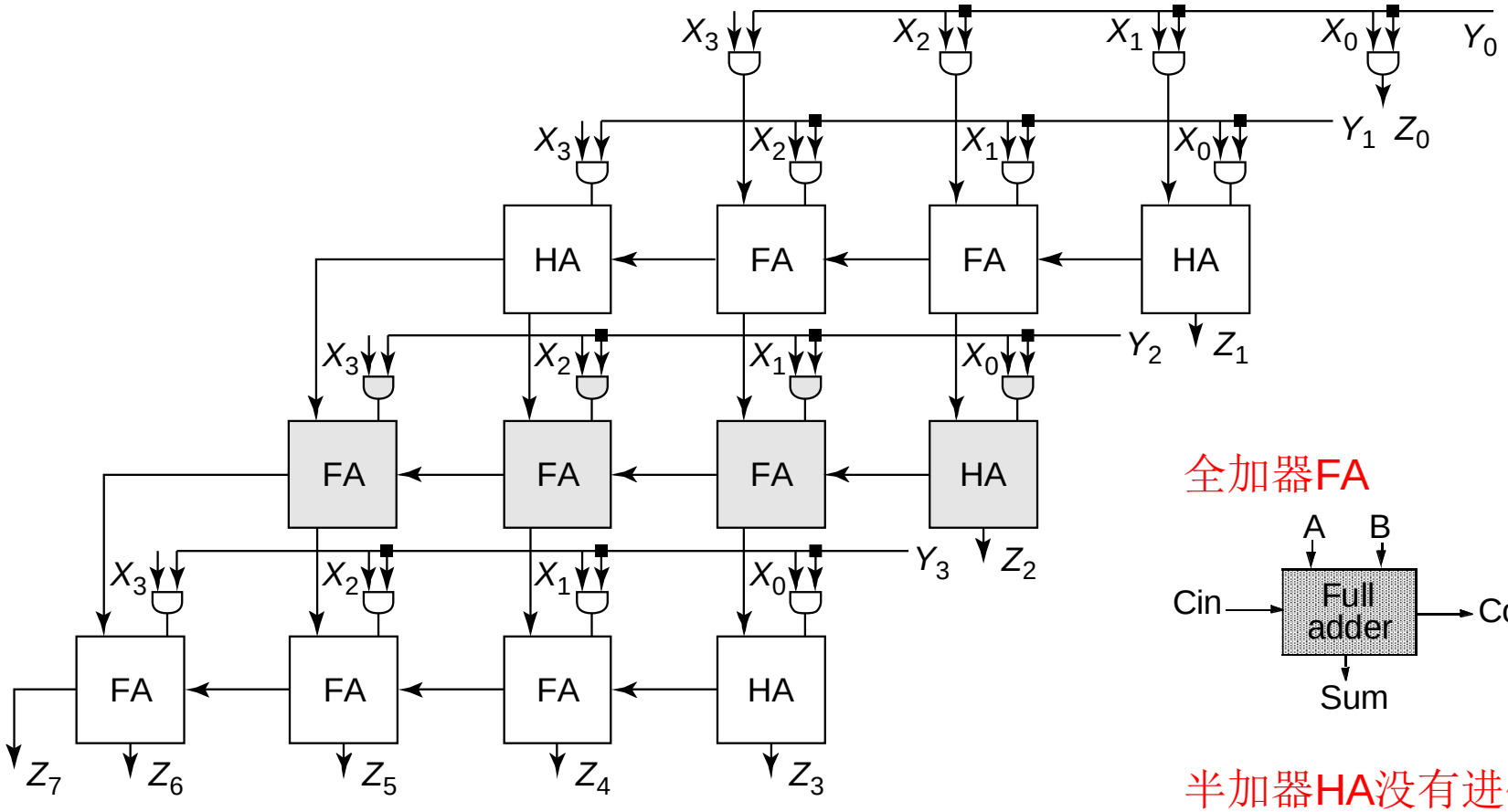


# of bits in P = # of bits in M + # of bits in Q

2个N位数相乘，产生N个部分积，需要N-1次加法

整数/浮点数运算能力
快速乘法器的组合逻辑实现结构

- 4bit的乘法有3次加法运算



全加器FA

半加器HA没有进位

使用快速乘法器后，可认为加法和乘法的时间代价相同

整数/浮点数运算能力
1T FLOPS的成本：20年前的超级计算机

◆1996年，美国Sandia国家实验室研发了超级计算机“ASCI Red”，浮点运算性能首次突破1TFLOPS。起初包含7264个计算节点、1212GB分布式内存和12.5TB磁盘存储容量。

◆该机器的原型使用的是Pentium Pro处理器（主频200MHz），后升级到Pentium II OverDrive处理器（主频333MHz）。升级后的系统拥有9632个处理器。ASCI Red超级计算机由104个机柜组成，占地面积达到了230平方米。

2015年6月全球十大超级计算机排名第一的是天河II号广州超算中心，54,902.4TFLOPS

整数/浮点数运算能力
1T FLOPS的成本：当代的独立显卡

- 2013年10月，AMD发布Radeon R9-290X显卡。拥有5.6T Flops的单精度浮点运算能力，理论上拥有1.4T的双精度浮点运算能力。
  - 讯景R9-290X显卡，￥2499.00 (京东，2015.09)
- 2015年6月，AMD发布Radeon R9-390X，8.6T Flops
  - 华硕R9-390X显卡，￥3599.00(京东，2015.09)
- 2014年9月， NVIDIA发布GTX 980，拥有4.6T Flops的单精度浮点运算能力
  - 华硕GTX 980显卡，￥4099.00(京东，2015.09)

整数/浮点数运算能力
1T FLOPS的成本：当代的集成显卡

- 2015年8月，Intel发布第6代酷睿i7-6700k集成了显示核心Intel® HD Graphics 530，HD 530(@24EU)理论上单精度浮点运算能力为441.6G FLOPS。
  - ￥2799.00 (京东，2015.09)
- 2015年6月，AMD发布Kaveri系列APU A10 7870k，集成了显示核心Radeon R7，该核心理论上单精度浮点运算能力为 806G Flops
  - ￥899.00(京东，2015.09)

CPU的超越函数指令

- 初等函数包括代数函数和超越函数。
- 代数函数指含加、减、乘、除和开方等基本运算符的数学函数，包括多项式、平方根函数等。
- 超越函数是“超出”代数函数范围的函数，指的是不满足任何以变量的多项式作为系数的多项式方程的函数，即变量之间的关系式不能用有限次的加、减、乘、除、乘方、开方这一类简单运算表示的函数。例如指数函数，对数函数，三角函数，反三角函数就属于超越函数。

CPU的超越函数指令
超越函数的近似计算方法

- 查表法
- 级数近似

$$f(x)=\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots+(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}+\ldots$$

$$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}-\ldots$$

←n的取值和需要的精度有关系

```
x=pi/4   = 0.78539816339744830961566084581988
-x^3/3!  = -0.08074551218828078170696957048724
x^5/5!   = 0.00249039457019272016001579842157
-x^7/7! =- 0.00003657620418217725078660518698
sin45=sin(pi/4)= 0.707106469575178070817920468567
```

$$e^x=\sum_{k=1}\frac{x^k}{k!}$$

- CORDIC ←最广泛
  - Many older systems with **integer-only CPUs** have implemented CORDIC to varying extents as part of their IEEE Floating Point libraries. As most modern general-purpose CPUs have floating-point registers with common operations such as add, subtract, multiply, divide, sin, cos, square root, log10, natural log, the need to implement CORDIC in them with software is nearly non-existent. Only microcontroller or special safety and time-constrained software applications would need to consider using CORDIC.

CORDIC algorithm

◆The modern CORDIC algorithm was first described in 1959 by Jack E. Volder. John Stephen Walther at Hewlett-Packard further generalized the algorithm, allowing it to calculate hyperbolic and exponential functions, logarithms, multiplications, divisions, and square roots.

◆坐标旋转数字计算机CORDIC(COordinate Rotation DIgital Computer)算法，通过**移位**和**加减**运算，能递归计算常用函数值，如Sin，Cos，Sinh，Cosh等函数，由J. Volder于1959年提出，首先用于导航系统，使得矢量的旋转和定向运算不需要做查三角函数表、乘法、开方及反三角函数等复杂运算。J. Walther在1974年用它研究了一种能计算出多种超越函数的统一算法。

http://en.wikipedia.org/wiki/CORDIC

CORDIC
(COordinate Rotation DIgital Computer)

- In early 1956 the aeroelectronics department of Convair, Fort Worth, was given the task of determining the feasibility of **replacing** the **analog** computer-driven navigation system of the B-58 bomber with a **digital** computer.
- Most navigation system specialists agreed that the existing B-58 navigation computer was an ingenuous device utilizing analog resolvers to compute, in real time, the complex trigonometric relationships necessary for navigation over a spherical earth. Each resolver was capable of performing either a rotation of input coordinates or inversely determining the magnitude and angle of the vector defined by the input coordinates—also called vectoring.

The Birth of CORDIC, JACK E. VOLDER,
Journal of VLSI Signal Processing 25, 101–105, 2000

CORDIC计算sin cos示意

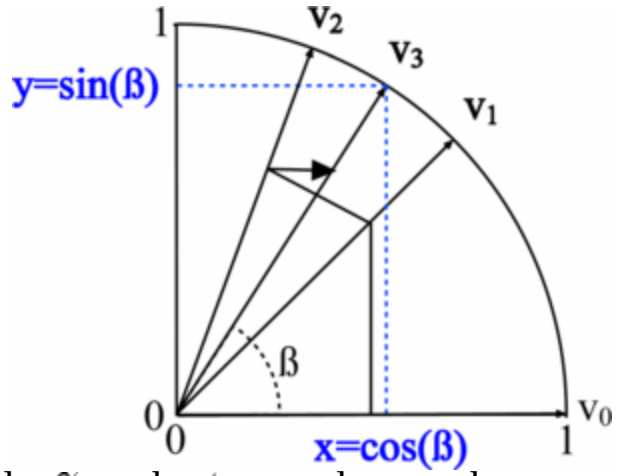
单位向量每次旋转γ_i角度，将sin计算转换为arctanγ_i的计算

$$v_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad v_i = R_i v_{i-1} \qquad R_i = \begin{bmatrix} \cos\gamma_i & -\sin\gamma_i \\ \sin\gamma_i & \cos\gamma_i \end{bmatrix}$$

$$\cos\alpha = \frac{1}{\sqrt{1+\tan^2\alpha}} \qquad \sin\alpha = \frac{\tan\alpha}{\sqrt{1+\tan^2\alpha}}$$

$$R_i = \frac{1}{\sqrt{1+\tan^2\gamma_i}} \begin{bmatrix} 1 & -\tan\gamma_i \\ \tan\gamma_i & 1 \end{bmatrix}$$

$$v_i = \frac{1}{\sqrt{1+\tan^2\gamma_i}} \begin{bmatrix} 1 & -\tan\gamma_i \\ \tan\gamma_i & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$



Restricting the angles γ_i so that $\tan\gamma_i$ takes on the values $\pm 2^{-i}$ the multiplication with the tangent can be replaced by a division by a power of two, which is done in digital computer hardware using a bit shift.

$$v_i = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \qquad K_i = \frac{1}{\sqrt{1+2^{-2i}}}$$

$$\beta_i = \beta_{i-1} - \sigma_i \gamma_i. \qquad \gamma_i = \arctan 2^{-i},$$

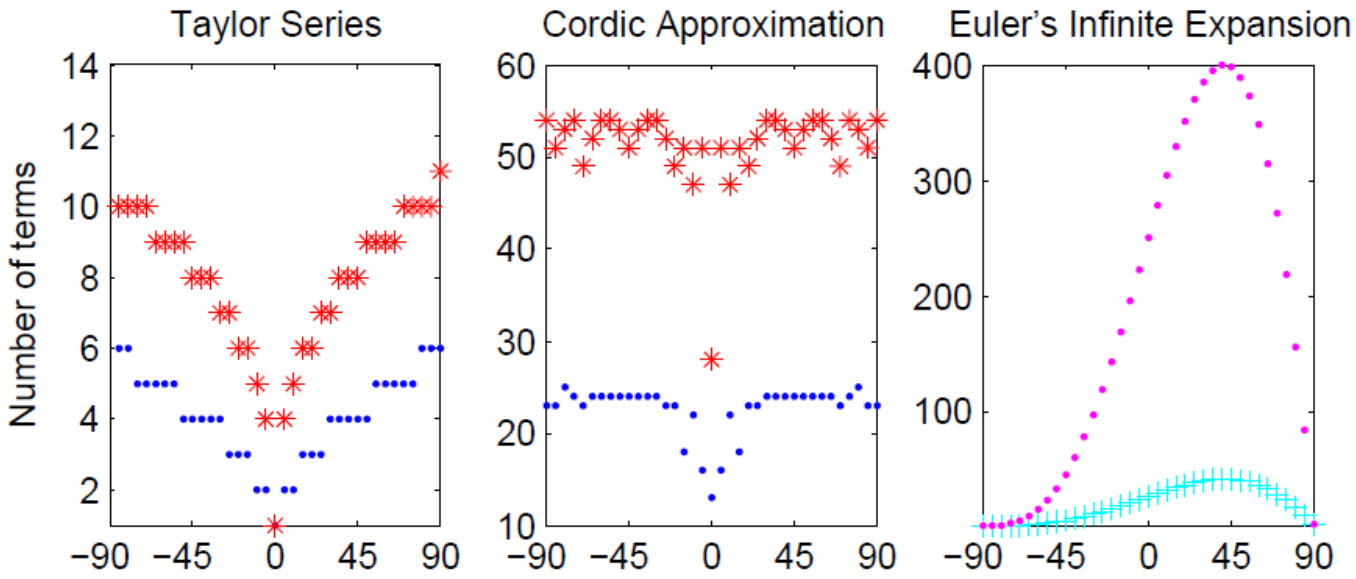
预先计算好

σ_i can have the values of -1 or 1 and is used to determine the direction of the rotation

CORDIC can be used to calculate a number of different functions. This explanation shows how to use CORDIC in *rotation mode* to calculate sine and cosine of an angle, and assumes the desired angle is given in radians and represented in a fixed point format. To determine the sine or cosine for an angle , the *y* or *x* coordinate of a point on the unit circle corresponding to the desired angle must be found.

cos(x)三种方法计算项的对比

The Taylor series requires more terms as x moves away from 0, while the Cordic expansion is not dependent on the value of x. Euler's infinite expansion is far worse.



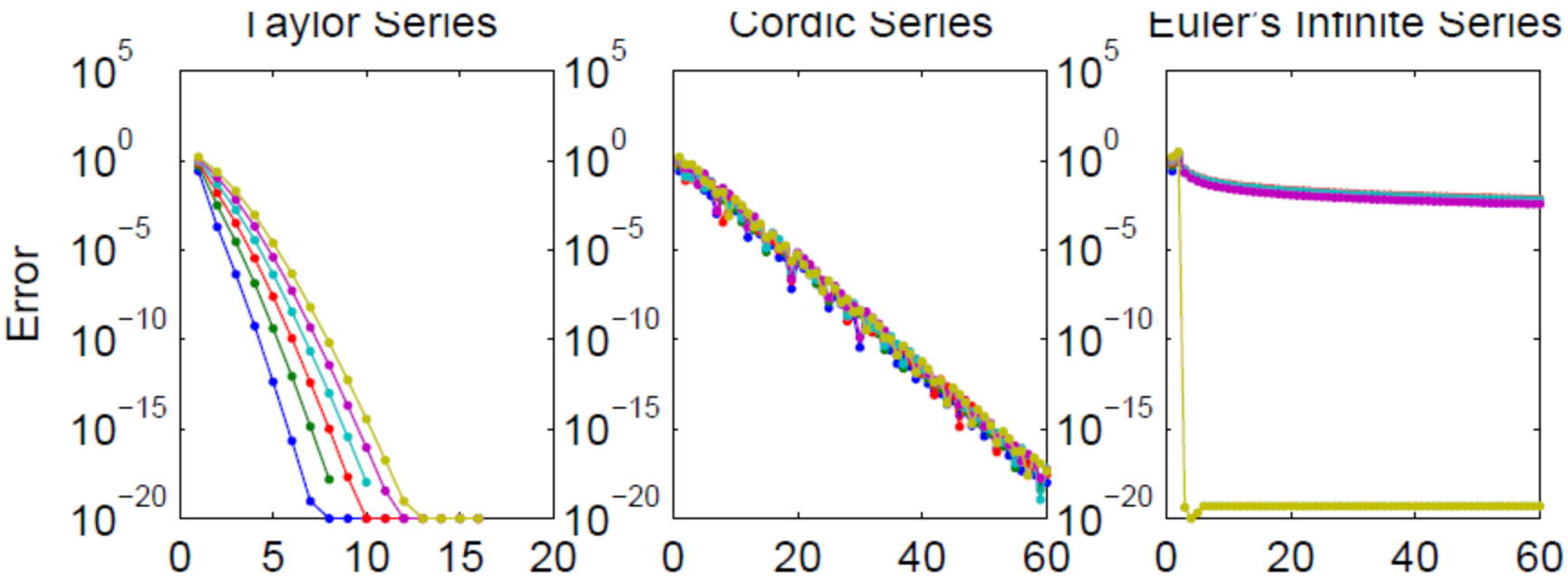
$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

蓝色：达到单精度
红色：达到双精度

$$\sin x = \prod_{n-1}^{\infty} \frac{(\pi n - x)\cdot(\pi n + x)}{\pi n \cdot \pi n} = x \cdot \prod_{n-1}^{\infty} \frac{1 - x^2}{\pi^2 n^2}$$

FAST TRIGONOMETRIC FUNCTIONS USING INTEL'S SSE2 INSTRUCTIONS
http://www.muddsnyder.com/pubs/fastSSE2.pdf

cos(x)三种方法收敛速度



Different values of x are used to demonstrate the dependence of convergence on the input value. The values chosen here are 0, 15, 30, 45, 60, 75, and 90 degrees. The Taylor series converges rapidly, yielding multiple digits per term. The Cordic series yields one bit per term. Euler's infinite series converges surprisingly slowly.

FAST TRIGONOMETRIC FUNCTIONS USING INTEL'S SSE2 INSTRUCTIONS
http://www.muddsnyder.com/pubs/fastSSE2.pdf

Intel i7中超越函数的运算速度
x87 Floating-point Instructions

FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	$2^x - 1$
FYL2X	$y*\log_2 x$
FYL2XP1	$y*\log_2(x+1)$

FADD	**1μops**
FSUB	3μops
FMUL	5μops
FDIV	32μops
FSQRT	58μops
FCOS	**119μops**
FSIN	119μops
FSINCOS	119μops
FPATAN	147μops
FPTAN	123μops
FYL2X	96μops
FYL2XP1	98μops

小结：多媒体计算需要什么？

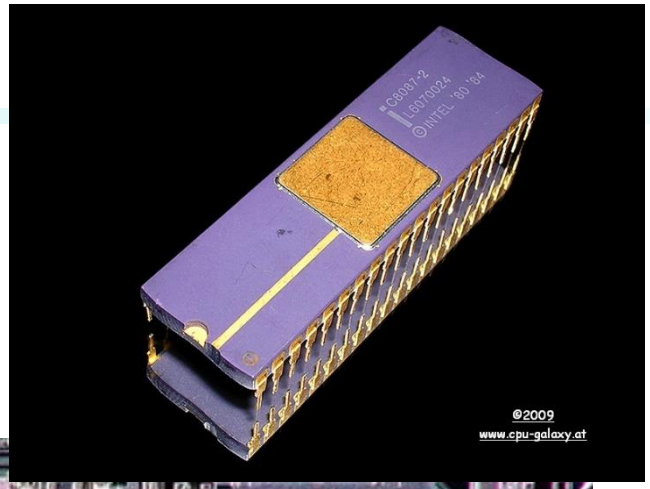
- 多媒体数据处理的特点
  - 计算密集、数据量大、可并行化
  - 整数、浮点数
  - 加减、乘除、超越函数
- 提高多媒体数据处理的可能途径
  - 提高CPU计算能力
    - 时间：主频
    - 空间：多个运算单元
    - 设计：Micro Architecture
  - 提高数据进出CPU的速度
    - I/O

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

协处理器集成到CPU内部

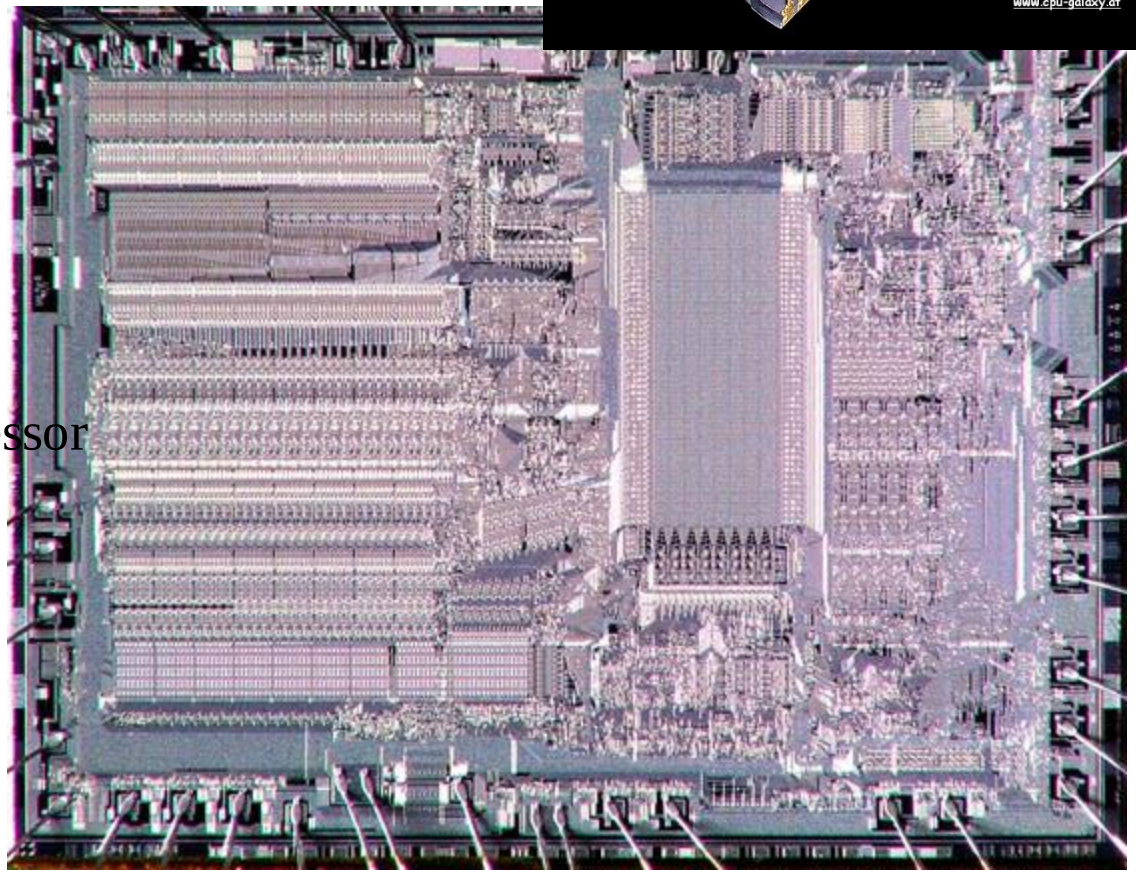
Intel 8087



第一款协处理器是配合给Intel 8086使用的8087。

Introduction date: 1978
Frequency:
8087 50 MHz
8087-1 100 MHz
8087-2 80 MHz
8087-3 40 MHz
8087-6 60 MHz

Technology: N Mos
Category: 16 Bit Math-Co Processor
Transistors: **45000**
Instructions: 68



http://www.cpu-galaxy.at/cpu/intel%20cpu/coprozessor/intel%208087%20section.htm

浮点数运算规则

$$x=2^{Ex}\cdot M_x \qquad y=2^{Ey}\cdot M_y$$

◆两浮点数进行加减，（1）对阶：必须使它们的阶码相等。（2）求和或求差。（3）规格化。

$$x\pm y=(M_x 2^{Ex-Ey}\pm M_y)2^{Ey},Ex\le E_y$$

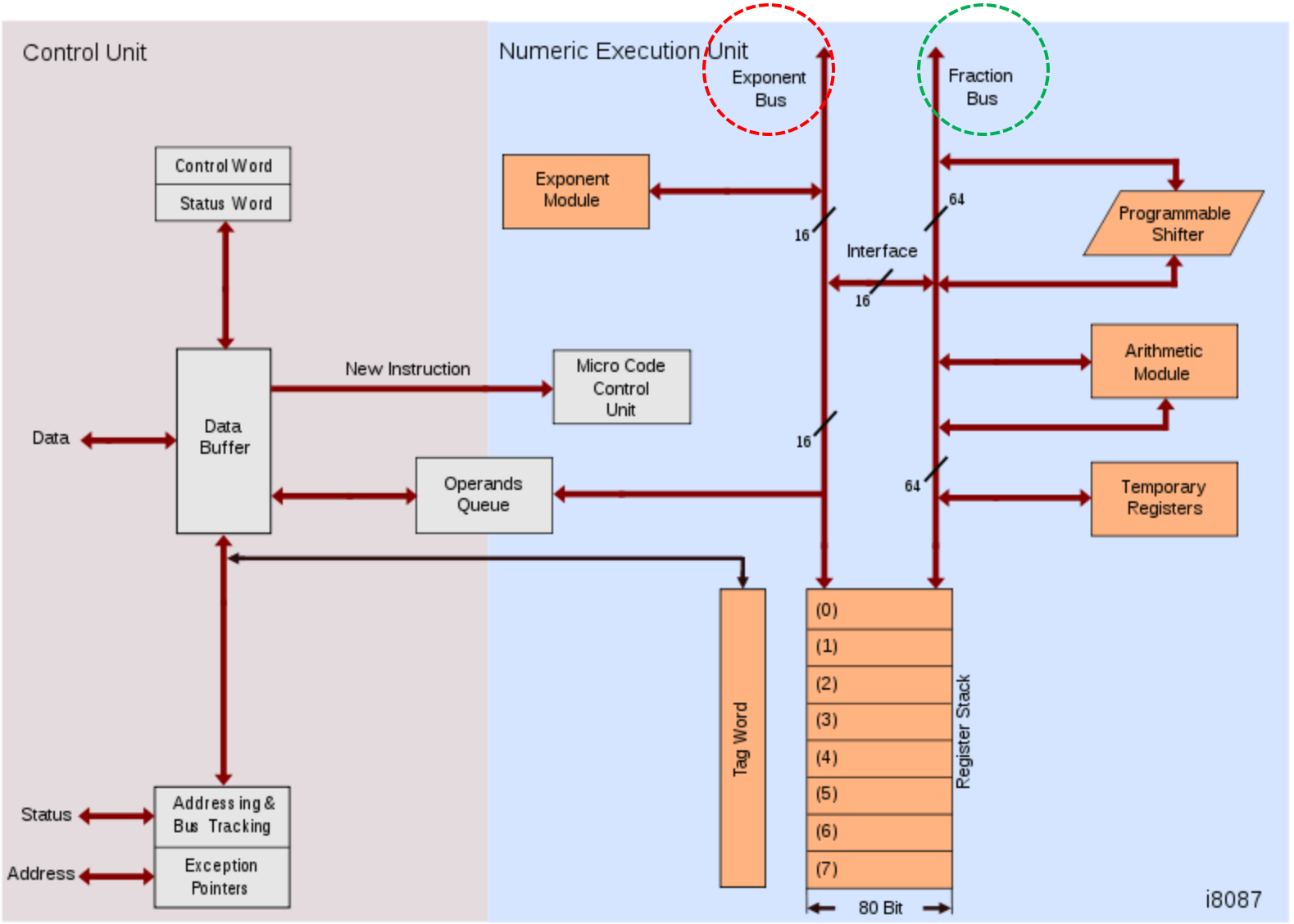
◆两浮点数相乘，其乘积的阶码为相乘两数的阶码之和，其乘积的尾数为相乘两数尾数之积。

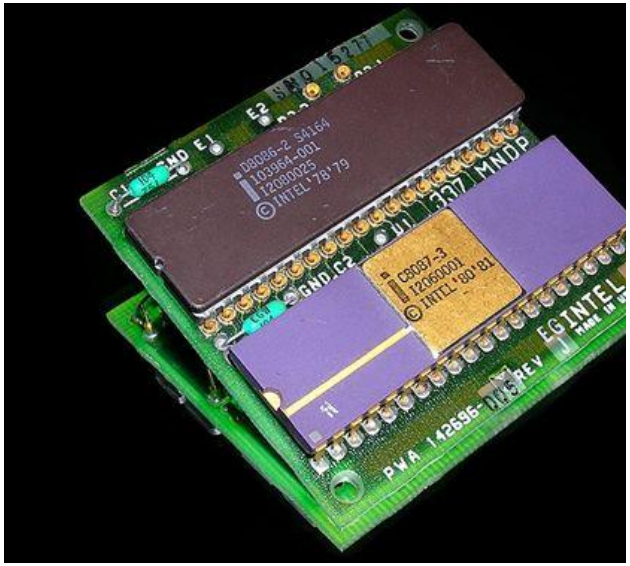
$$x\times y=2^{(Ex+Ey)}\cdot(M_x\times M_y)$$

◆两浮点数相除，商的阶码为被除数的阶码减去除数的阶码所得到的差，尾数为被除数的尾数除以除数的尾数所得的商。

$$x\div y=2^{(Ex-Ey)}\cdot(M_x\div M_y)$$

协处理器集成到CPU内部
8087 microarchitecture





1978年
8087
Transistors: **45000**
+
8086
Transistors: 约**6000**

1990年
Intel 80486DX - 具有FPU的i486

Table 3. Execution Times for Selected 8086/8087 Numeric Instructions and Corresponding 8086 Emulation

Floating Point Instruction	Approximate Execution Time (μs)	
	8086/8087 (8 MHz Clock)	8086 Emulation
Add/Subtract	10.6	1000
Multiply (Single Precision)	11.9	1000
Multiply (Extended Precision)	16.9	1312
Divide	24.4	2000
Compare	−5.6	812
Load (Double Precision)	−6.3	1062
Store (Double Precision)	13.1	750
Square Root	22.5	12250
Tangent	56.3	8125
Exponentiation	62.5	10687

协处理器集成到CPU内部

386+387

图示387协处理器与386微处理器并行连接，可以构成一个高效的386处理器系统，而缺少了协处理器的单独386芯片浮点运算能力非常有限。实际上387芯片相对于386的改变就是增加了8个80位的浮点寄存器，以及16位的控制寄存器、状态寄存器和标志寄存器。这样387协处理器就为386处理器扩充了七十多条指令和多种数据类型，使得386处理器的浮点也能够遵循IEEE754浮点标准。

386处理器与387协处理器

协处理器集成到CPU内部
8087 vs. 387

Intel387™ DX MATH COPROCESSOR

- High Performance 80-Bit Internal Architecture
- Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
- Expands Intel386™ DX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- Directly Extends Intel386™ DX CPU Instruction Set to Include Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data Types
- Upward Object-Code Compatible from 8087 and 80287
- Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM
- Built-In Exception Handling
- Operates Independently of Real, Protected and Virtual-8086 Modes of the Intel386™ DX Microprocessor
- Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
- Available in 68-Pin PGA Package
- One Version Supports 16 MHz–33 MHz Speeds

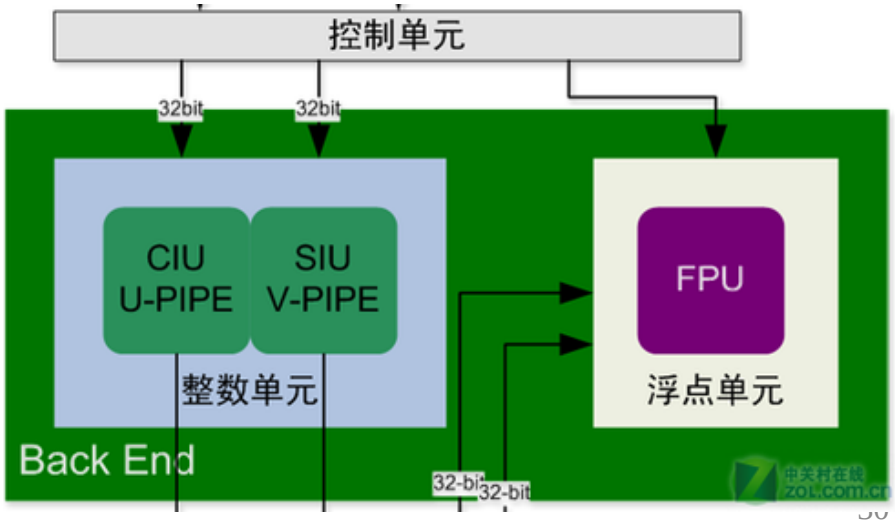
http://www.nj7p.org/Manuals/PDFs/Intel/240448-005.PDF

协处理器集成到CPU内部

486DX

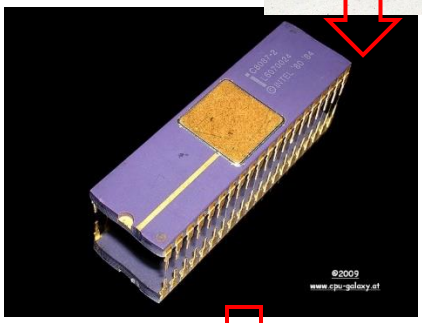
1989年，Intel发布了486处理器。486最初分为有数学协处理器的486DX和无数学协处理器的486SX两种，后来486DX已经完全替代了SX系列成为主流选择，历史的车轮让协处理器概念逐步淡出我们的视野。486DX整合了协处理器，协处理器的概念从此消失而变为CPU内部的FPU（浮点处理器）。

FPU: floating point unit



小结：协处理器集成到CPU内部

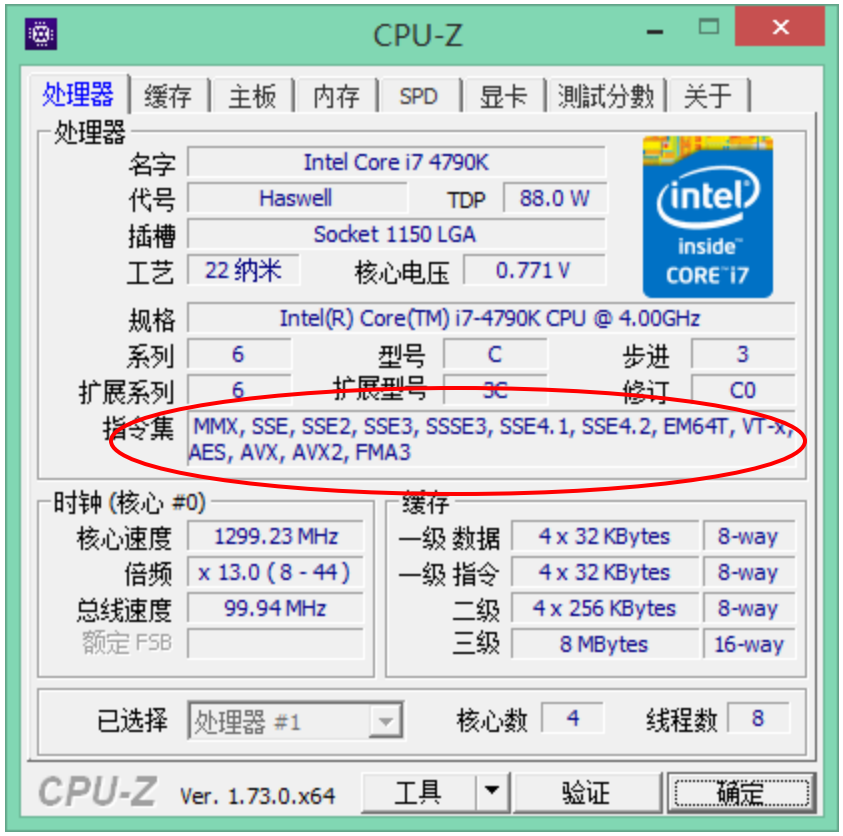
- 1971年，Intel 4004
  - 第一片微处理器
- 1978年，8087
  - 16bits Math-Co Processor
- 1989年，496DX
  - 32bits FPU: floating point unit



专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

CPU的扩展指令集



CPU都有一个基本的指令集，对于同一个档次的CPU，其基本指令集都差不多，但是为了提升某方面的性能，增加一些特殊的指令和硬件功能，使计算机处理信息的速度得到很大提高，这些新增加的指令就构成了扩展指令集。

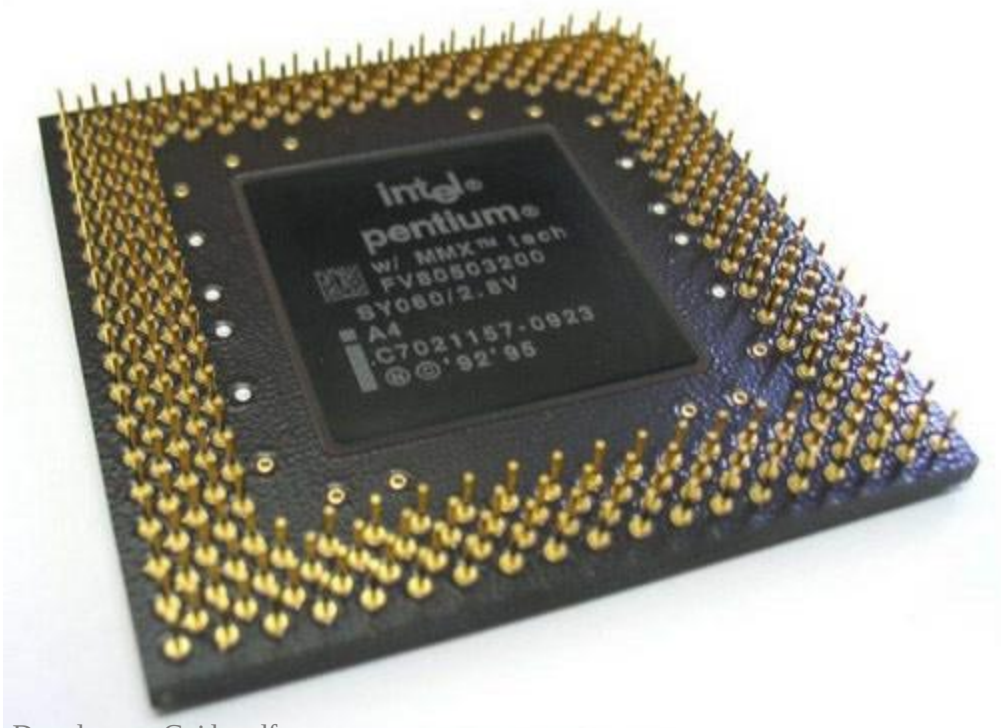
常见指令集扩展用于增强CPU的多媒体、图形图象、Internet等的处理能力。

CPU的扩展指令集

SIMD (Single Instruction Multiple Data)的出现

第一套扩展指令集MMX（Multi Media eXtension）发布于1997年，一共57条指令。MMX是SIMD（单指令多数据）技术的一个最简单的部分，它只能对整数（BYTE、WORD、DWORD、QWORD）进行操作，而且提供的指令也有限。

- MMX指令集增加了57条新的操作码和一个新的64位四字数据类型
- 增加了八个新的64位MMX寄存器，每个寄存器可按名称MM0-MM7直接访问。

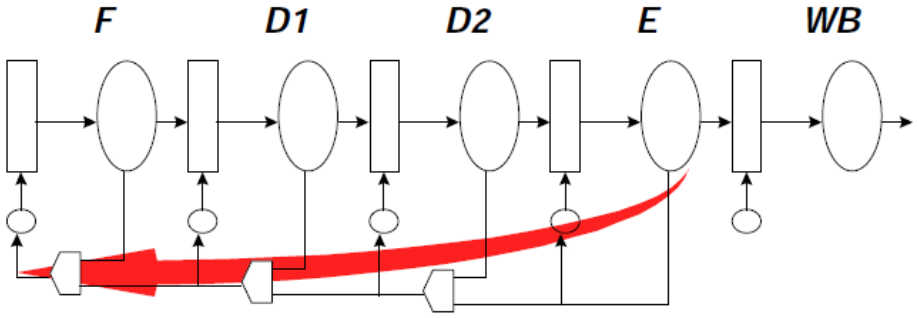


https://software.intel.com/sites/default/files/m/8/0/a/MMX_Manual_Tech_Developers_Guide.pdf

MMX的57条多媒体指令的真实面貌

- 操作数的比特宽度：64bits
- 流水线的变化应对2处瓶颈
  - decoder
  - data cache access

Pentium pipeline



Pentium with MMX technology pipeline

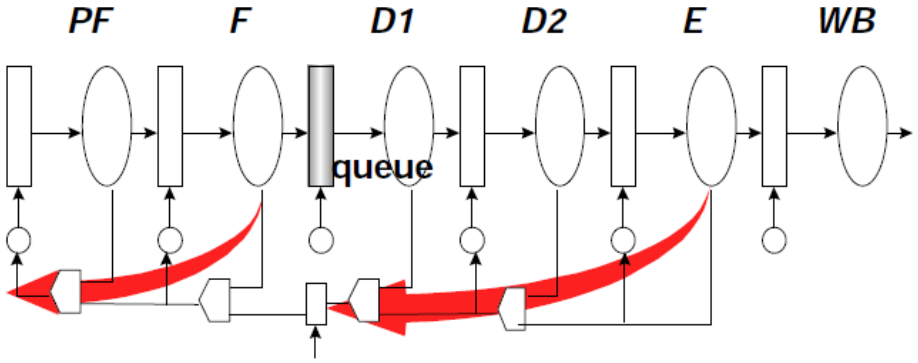


Table A-1. Intel Architecture MMXTM Instruction Set			
Packed Arithmetic	**Wrap Around**	**Signed Sat**	**Unsigned Sat**
Addition	PADD	PADDS	PADDUS
Subtraction	PSUB	PSUBS	PSUBUS
Multiplication	PMULL/H		
Multiply & add	PMADD		
Shift right Arithmetic	PSRA		
Compare	PCMPcc		
Conversions	**Regular**	**Signed Sat**	**Unsigned Sat**
Pack		PACKSS	PACKUS
Unpack	PUNPCKL/H		
Logical Operations	**Packed**	**Full 64-bit**	
And		PAND	
And not		PANDN	
Or		POR	
Exclusive or		PXOR	
Shift left	PSLL	PSLL	
Shift right	PSRL	PSRL	
Transfers and Memory Operations	**32-bit**	**64-bit**	
Register-register move	MOVD	MOVQ	
Load from memory	MOVD	MOVQ	
Store to memory	MOVD	MOVQ	
Miscellaneous			
Empty multimedia state	EMMS		

CPU的扩展指令集

CPU扩展指令集历程

多媒体指令集本质是一种SIMD数据处理方式

•**MMX 指令**：Intel 在 1996 年推出的第 1 代 **SIMD**（Single Instruction Multiple Data）指令集，随后 AMD支持MMX。
•**3DNow! 指令**：1998 年 AMD 首发推出了 21 条自己的 SIMD 指令集，3DNow! 的性能要优于 MMX 指令，使用在 AMD K6-2 处理器上。随后 AMD 在 1999 年 6 月发布的 Athlon 处理器上使用了增强版的 3DNow! 指令（3DNow!+）。
•**SSE 指令**：1999 年 Intel 推出了第 1 代的 SSE（Streaming SIMD Extensions）指令以回击 AMD 的 3DNow! 指令，使用在 Pentium III 处理器上。增加了8个**128位寄存器**XMM0-XMM7。随后 AMD 在 2001 年 10 月 发布 的 Athlon XP 处理器上首次加入了 SSE 指令集。
•**SSE2 指令**：Intel 在 2001 年推出，用在 Pentium 4上，AMD 在 2003 年推出的 Athlon 64 和 Opteron 上加入 SSE2。
•**x86-64 指令**：2003 年 AMD 推出了第 8 代名为 K8 的 microarchitecture，实现了 x86-64 架构，支持 64位 的扩展技术。AMD 将自己的 x86-64 架构实现称为 AMD64 架构。Intel 最终在 2004 年发布的 Prescott 微架构的 Pentium 4 处理器上实现 x86-64 扩展技术，称为 **EM64T** 技术，兼容于 AMD64 架构。这是 Intel 唯一的一次追随 AMD。
•**SSE3 指令**：Intel 在 2004 年推出，用在 Prescott 微架构的 Pentium 4 上， 2005 年 AMD 在 Athlon 64加入 SSE3 支持。
•**SSSE3 指令**：SSSE3 指令是对 SSE3 指令的补充，新增了 16 条指令，在最后的一版 Prescott 微架构代号为 Tejas 的 Pentium 4 上首次加入 SSSE3 指令，以及 2006 年的 Core 微架构的处理器上开始加入 SSSE3 指令。在 AMD 阵营已支持。
•**SSE4.1** 指令：Intel 在 2007 年 11 月发布的 Penryn 微架构的处理器上加入了 SSE4.1 指令，SSE4.1 指令共 47 条。在AMD阵营中，目前发布的处理器不支持 SSE4.1 指令，即将发布的 Bulldozer 微架构的处理器将支持 SSE4.1 指令。
•**SSE4a 指令**：AMD 在 2007 年在 K10 微架构的处理器加入了 SSE4A 指令集，SSE4A 只有 4 条。在 K10 微架构处理器上还加入 POPCNT 与 LZCNT 指令
•**SSE4.2 指令**：Intel 在 2008 年 11 月发布的 Nehalem 微架构的 Core i7 处理器上加入 SSE4.2 指令共 7 条。在 AMD 阵营中，目前发布的处理器不支持 SSE4.2 指令，即将发布的 Bulldozer 微架构的处理器将支持 SSE4.2 指令
•**SSE5 指令**：SSE5 是一个纸面上的指令集，并没有最终实现，AMD 在 2007 年 8 月公布 SSE5 指令集规范。
•**AVX 指令**：2008 年 3 月 Intel 发布了 AVX（Advanced Vector Extensions）指令集， 128bits寄存器扩展为**256bits**。首次在 Sandy Bridge 微架构的 Core i7/i5/i3 处理器上使用。AMD 将在 Bulldozer 微架构的处理器上加入 AVX 指令的支持。
•**AES 指令**：2008 年 3 月 Intel 发布了 AES（Advanced **Encryption** Standard）指令，使用在 Westmere 微架构的 Core i7/i5 处理器上。AMD 将在 Bulldozer 微架构的处理器上使用。
•**FMA 指令**：FMA 指令是 AVX 指令集中的一部分，Intel 将在 2013 年的 Haswell 微架构处理器上使用。
•**XOP，FMA4 以及 CVT16 指令**：AMD 在 2009 年 5 月发布了 XOP，FMA4 以及 CVT16 指令，这些指令集取代了 SSE5 指令，在原有的 SSE5 指令基础上，使用了兼容 AVX 指令的设计方案重新进行了设计。

示例：YUVtoRGB一般的C编程实现

```
void YUV420p_to_RGB24(unsigned char *yuv420[3], unsig
ned char *rgb24, int width, int height)  {
    int R,G,B,Y,U,V;
    int x,y;
    int nWidth = width>>1; //色度信号宽度
    for (y=0;y<height;y++)     {
        for (x=0;x<width;x++)          {
            Y = *(yuv420[0] + y*width + x);
            U = *(yuv420[1] + ((y>>1)*nWidth) + (x>>1));
            V = *(yuv420[2] + ((y>>1)*nWidth) + (x>>1));
            R = Y + 1.402*(V-128);
            G = Y - 0.34414*(U-128)  - 0.71414*(V-128);
            B = Y + 1.772*(U-128);
            *(rgb24 + ((height-y-1)*width + x)*3) = B;
            *(rgb24 + ((height-y-1)*width + x)*3 + 1) = G;
            *(rgb24 + ((height-y-1)*width + x)*3 + 2) = R;
        }
    }
}
```

示例：YUVtoRGB SSE指令实现

- SSE2中寄存器xmm0-xmm7 可保存128 位的数据
- pmulhw 指令对xmm0内的数据做有符号乘
- psraw 指令对寄存器内的数据做算术右移（除法）

方法1：浮点数系数用整数运算来近似

$$
\begin{aligned}
R &= Y + 1.407\,5(V-128)\\
&= Y + (360/256)(V-128)\\
&= Y + 360(V-128)/256\\
&= Y + (360(V-128)) >> 8
\end{aligned}
$$

图 5 计算 1 000帧所需时间

方法2：8个值成一组，用SSE的128bits指令计算

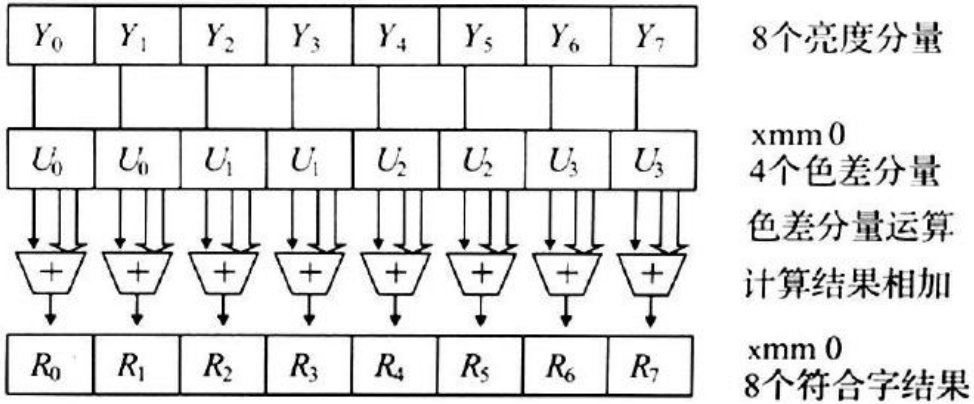


图 3 RGB格式红色分量 R 的 SSE2计算

基于SSE2的YUV与RGB色彩空间转换
中国图象图形学报， 2010-01-15

CPU的扩展指令集
Intel® Advanced Vector Extensions (Intel® AVX)

◆Advanced Vector Extensions (AVX) are extensions to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008 and first supported by Intel with the Sandy Bridge processor shipping in Q1 2011 and later on by AMD with the Bulldozer processor shipping in Q3 2011. AVX provides new features, new instructions and a new coding scheme.

◆AVX2 expands most integer commands to 256 bits and introduces FMA. AVX-512 expands AVX to 512-bit support utilizing a new EVEX prefix encoding proposed by Intel in July 2013 and first supported by Intel with the Knights Landing processor scheduled to ship in 2015.

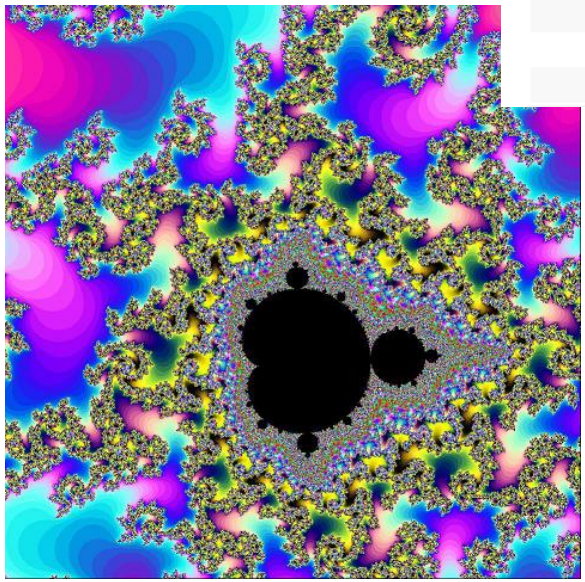
CPU的扩展指令集
Intel® Advanced Vector Extensions (Intel® AVX)

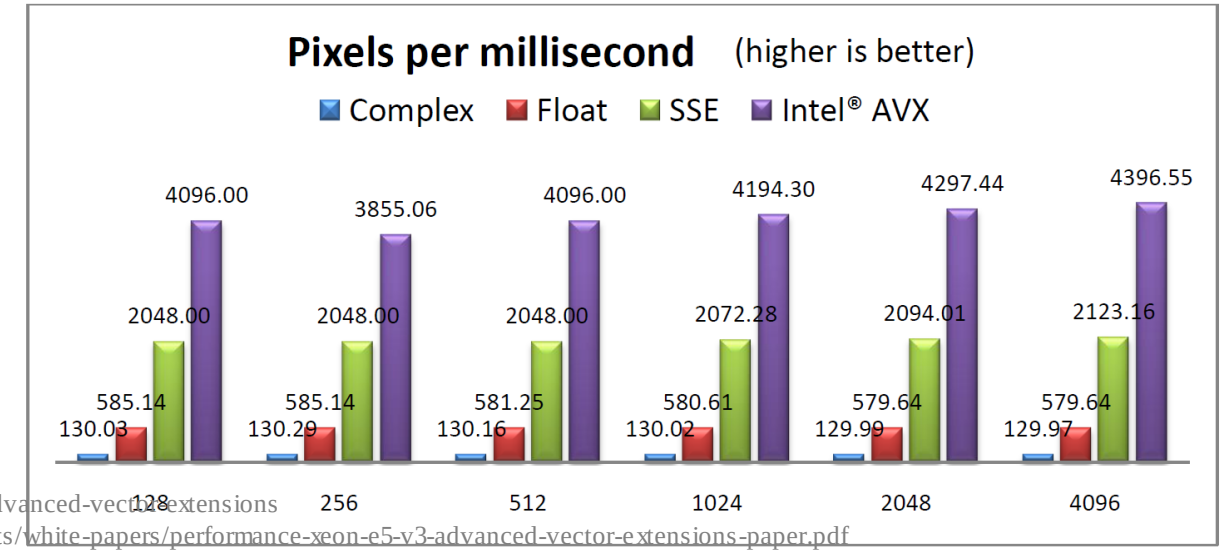
- The 128-bit SIMD registers have been expanded to 256 bits.
- Three-operand, nondestructive operations have been added.

Listing 3. *Mandelbrot Pseudocode*

```
z,p are complex numbers
for each point p on the complex plane
    z = 0
    for count = 0 to max_iterations
        if abs(z) > 2.0
            break
        z = z*z+p
    set color at p based on count reached
```

Mandelbrot set (0.29768, 0.48364) to (0.29778, 0.48354), with max iterations of 4096

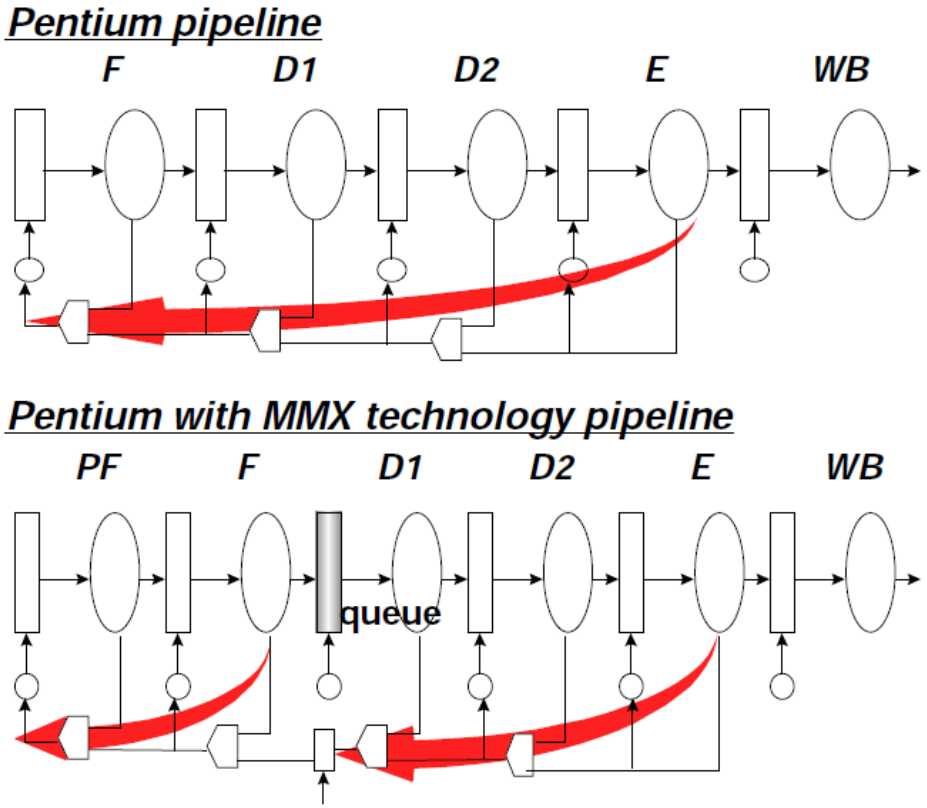




https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf

小结：CPU的扩展指令集

- 扩展指令集不仅仅是指令增加
- 操作数数目
- 操作数比特宽度
- 整数→浮点数
- 流水线

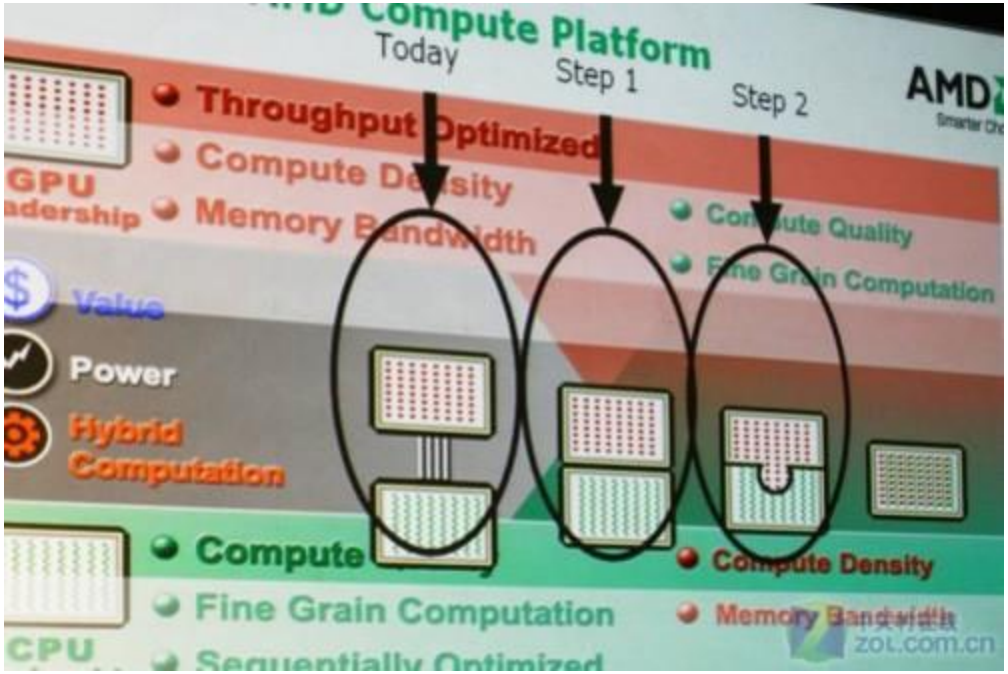


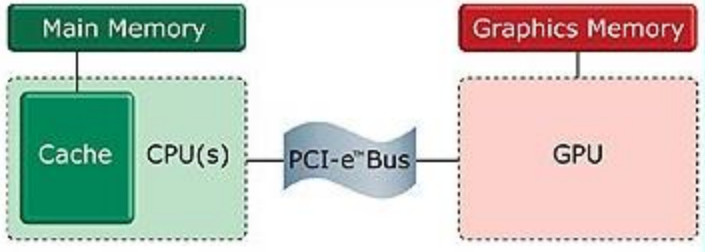
专题：多媒体计算→异构计算

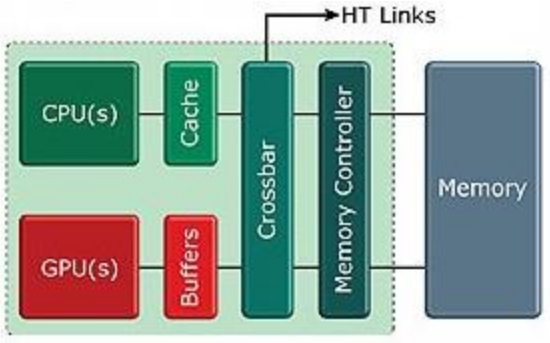
- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

APU(CPU+GPU)
CPU与GPU进入同一块芯片

◆2005年，在AMD收购ATI后发表代号为Fusion的研发计划，预计将AMD的CPU与ATI的GPU整合在一起，同时也将北桥芯片也一并纳入。



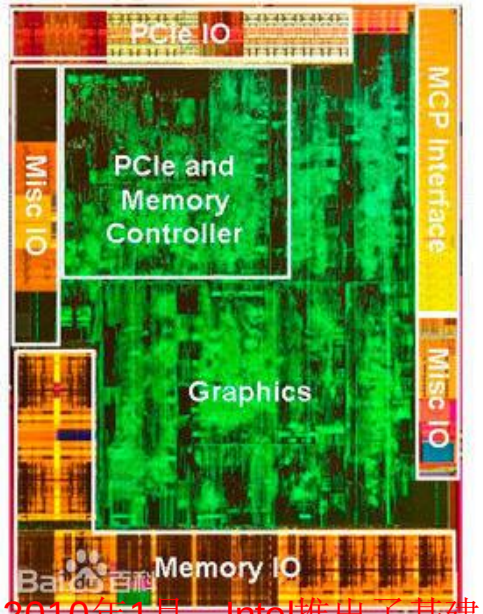


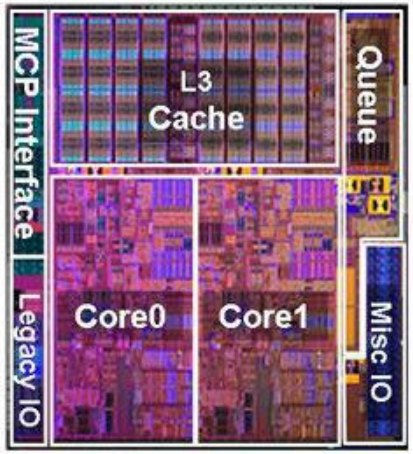


◆但是历史又一次向我们展示了它不可预测的一面，首先将GPU整合在CPU中的厂商，竟然是这个领域长期沉默的Intel。在2010年初，Intel带来了两款整合GPU的CPU系列产品——i3 500和 i5 600系列。

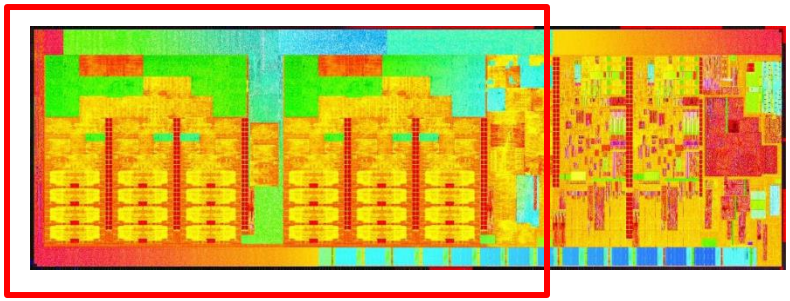
APU(CPU+GPU)
Intel Ivy Bridge之后CPU均集成GPU

在Intel推出HD Graphics以前，Intel的集成显示核心是集成于北桥芯片中，包括Intel Extreme Graphics和Intel GMA在内，均采用此种设计。后来Intel在推出Nehalem微架构(2010年)以后逐步推行单芯片组设计——北桥大部分集成于处理器上，小部分北桥的功能另外集成于剩下的南桥芯片内，是为Intel官方所谓的“PCH单芯片”（Platform Controller Hub）设计。同样，原来集成于北桥的显示核心移步至处理器上。





Intel® Core™ i5 with Iris graphics 6100



2010年1月，Intel推出了基建于Nehalem微架构，核心代号“Clarkdale”和“Arrandale”的处理器，其集成了HD Graphics的首款产品HD Graphics 1000

APU(CPU+GPU)
AMD Accelerated Processing Unit (APU)

◆The AMD Accelerated Processing Unit (APU), formerly known as Fusion, is a series of 64-bit microprocessors from AMD designed to act as a CPU and graphics accelerator (GPU) on a single chip.

- K10 architecture (2011): Llano ←第一款APU
- Bobcat architecture (2011): Ontario, Zacate, Desna, Hondo
- Piledriver architecture (2012): Trinity and Richland
- Jaguar architecture (2013): Kabini and Temash
- ARM server architecture (2014): Seattle
- Steamroller architecture (2014): Kaveri
- Puma architecture (2014): Beema and Mullins

海思Kirin 925

◆2014年9月，华为发布了新旗舰Mate 7，与之一同亮相的还有海思的新旗舰处理器Kirin 925。Kirin 925是四核心Cortex-A7+四核心Cortex-A15架构设计，主频1.8GHz，搭配Mali-T628MP4 GPU。

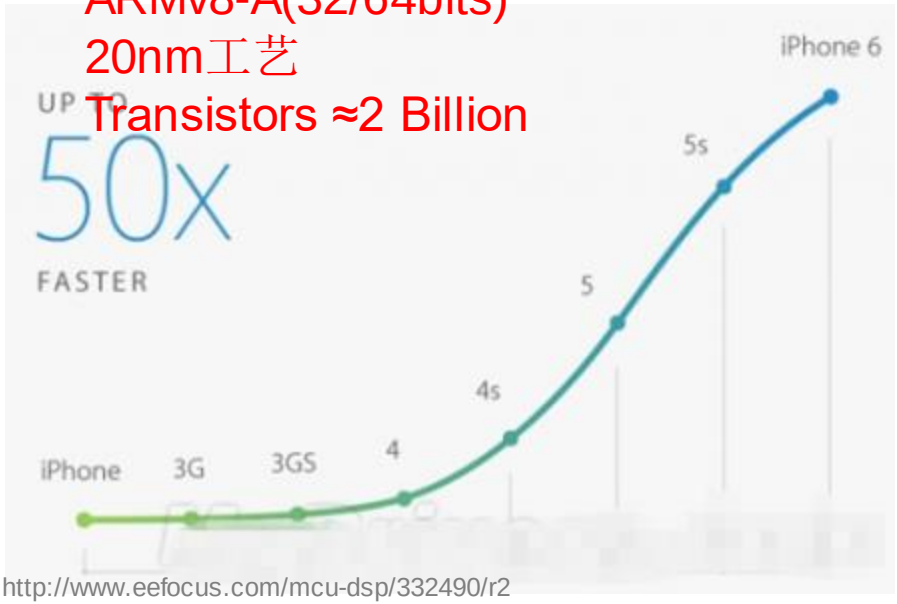
iPhone6 A8处理器 2014.09

芯片面积≈89mm²

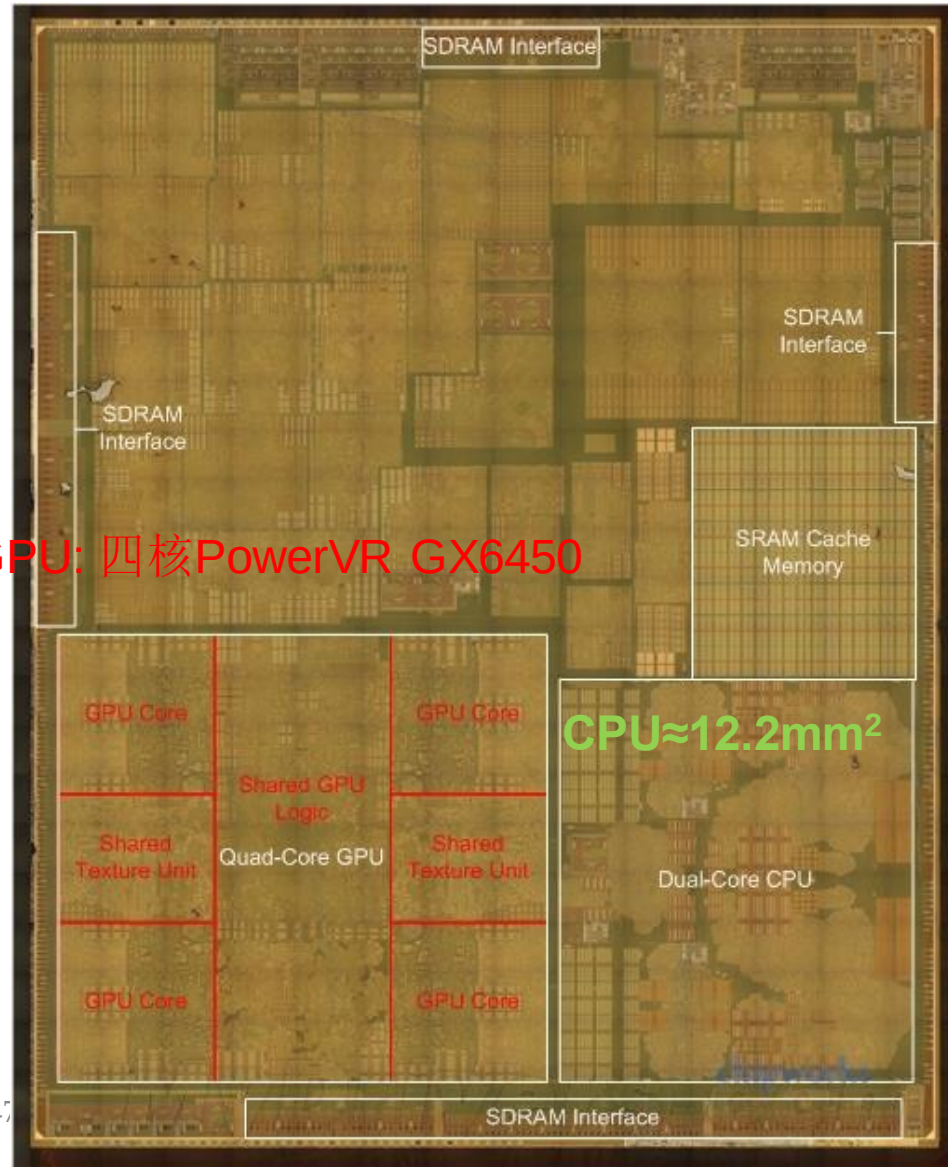
ARMv8-A(32/64bits)

20nm工艺

Transistors ≈2 Billion



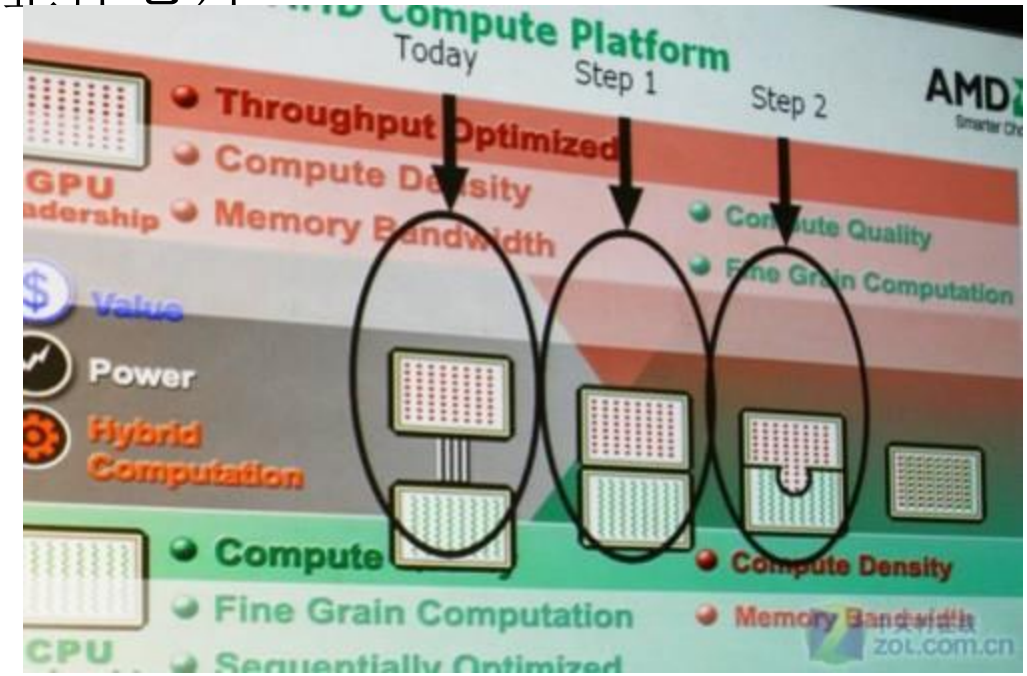
http://www.eefocus.com/mcu-dsp/332490/r2



GPU: 四核PowerVR GX6450

小结：CPU+GPU

- CPU集成GPU的三个阶段
  - GPU由扩展卡集成到主板上
  - GPU与CPU放置在同一硅衬底和封装中
  - GPU与CPU设计为整体芯片
- PC
  - 2010，Intel
  - 2011，AMD
- 智能终端

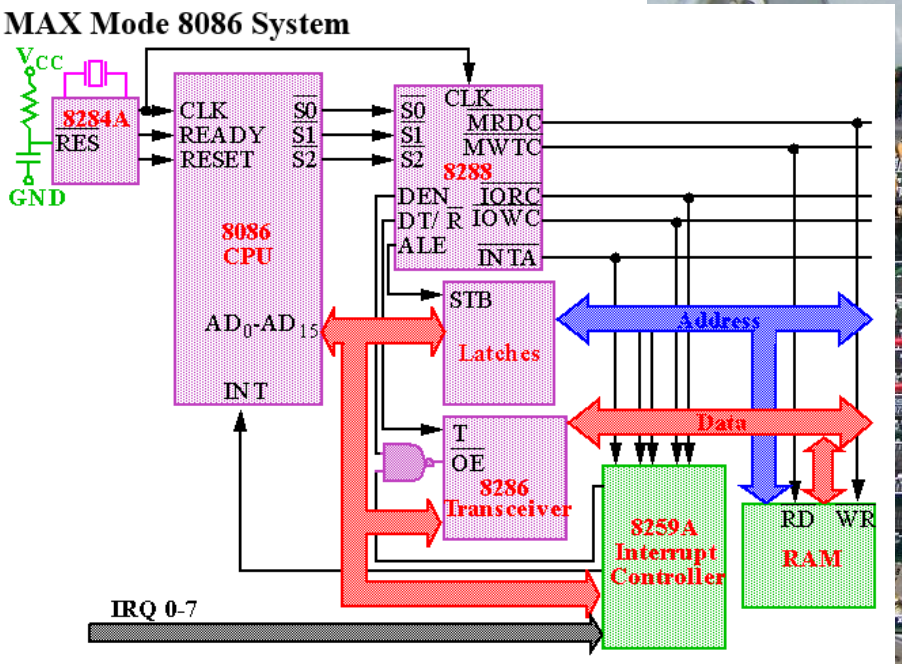


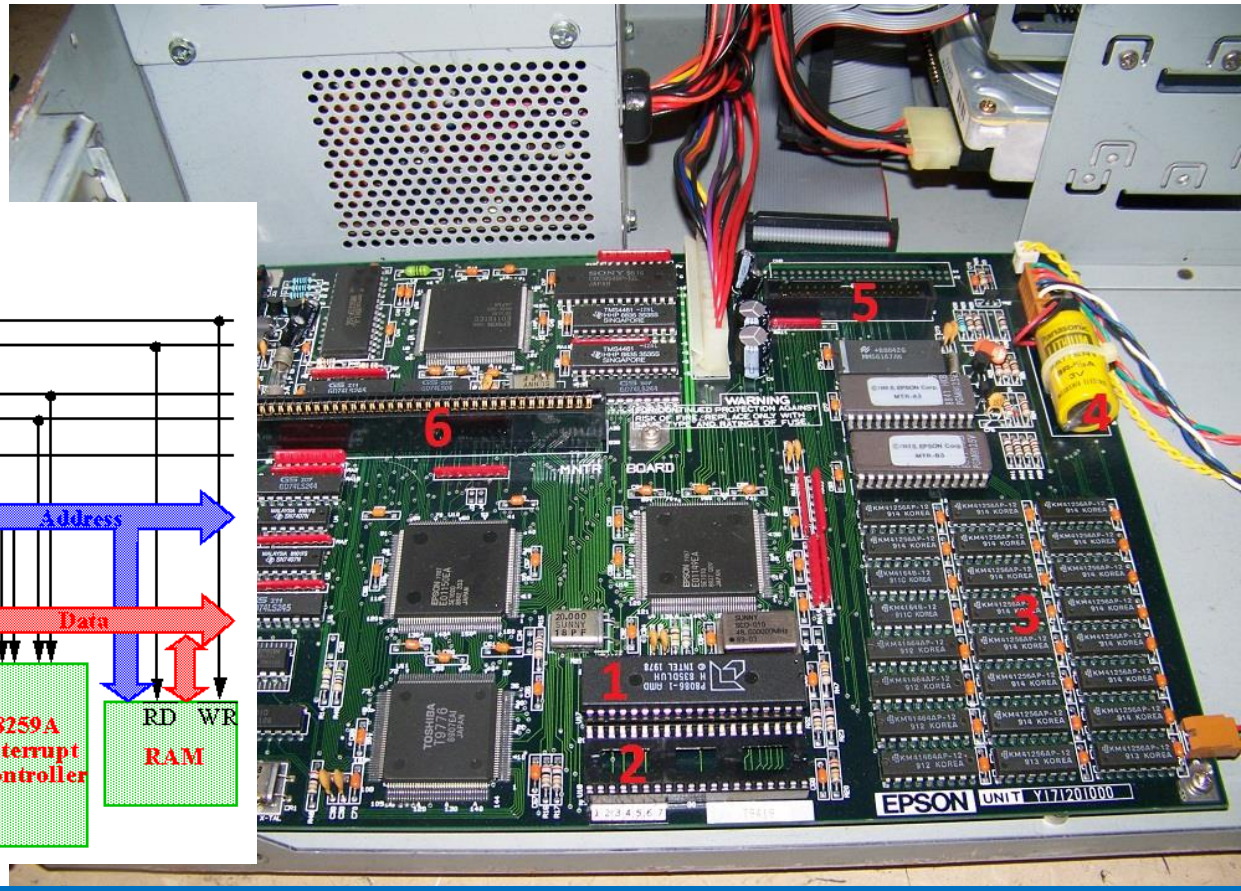
专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

PC主板芯片组背后的故事
8086时代的微型计算机

1) CPU – 8086 processor (Intel于1978年推出)
2) Co-Processor socket – optional 8087 math co-processor.
3) RAM – This is the built in 640kb of system RAM.
4) CMOS battery
5) Floppy connector
6) Riser slot –8-bit ISA cards

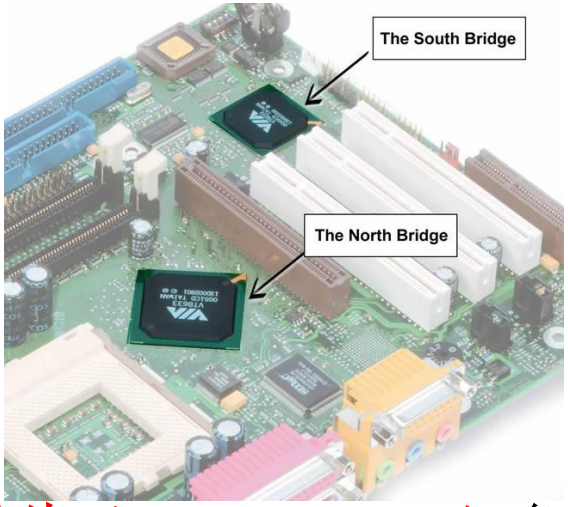




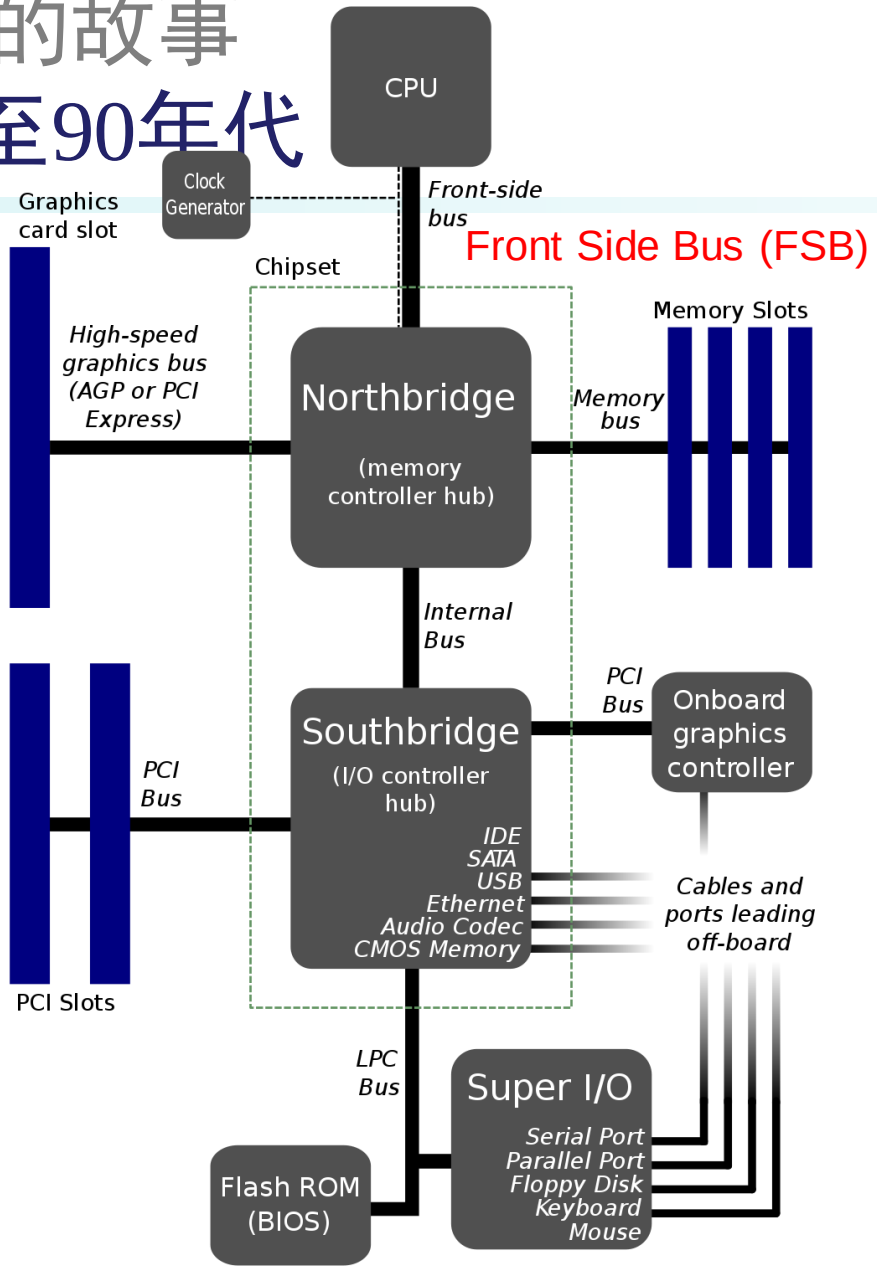
x86 PC诞生之初并没有专门的芯片组概念。

PC主板芯片组背后的故事
20世纪80年代后期至90年代

北桥芯片（North Bridge）负责与CPU的联系并控制内存，整合型芯片组的北桥芯片还集成了图形处理器。



南桥芯片（South Bridge）负责I/O总线之间的通信，如PCI总线、USB、LAN、ATA、SATA、音频控制器、键盘控制器、实时时钟控制器、高级电源管理等。



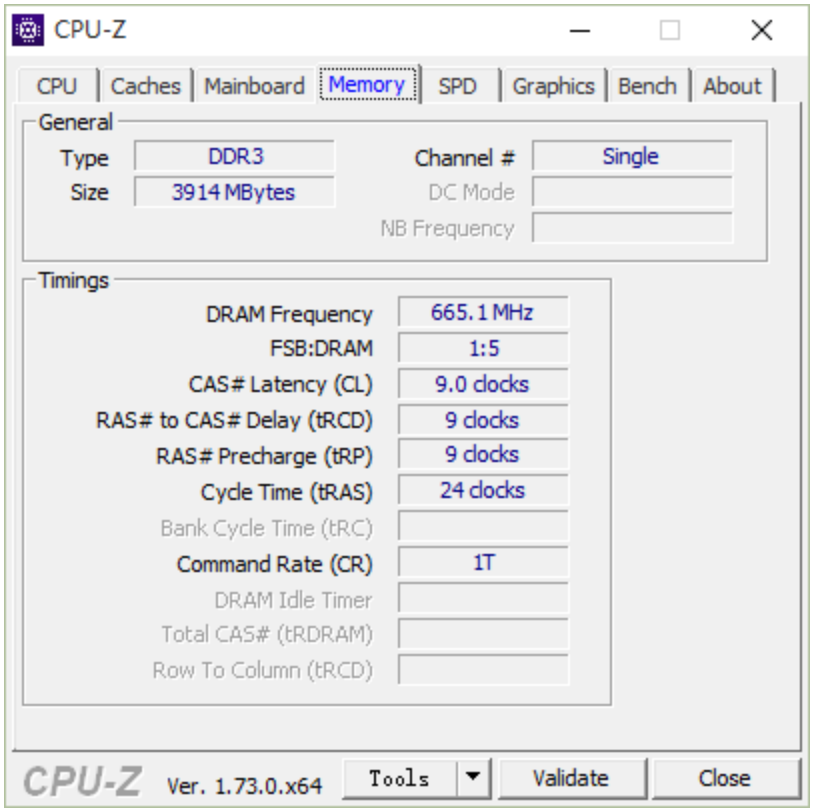
进入386时代后，“北桥”、“南桥”双芯片结构的芯片组正式确立。

PC主板芯片组背后的故事

主频达到瓶颈后：CPU集成内存控制器

传统的内存控制器位于主板芯片组的北桥芯片内，数据经由多级传输，数据延迟显然比较大从而影响计算机系统的整体性能。

Memory controllers contain the logic necessary to **read and write** to DRAM, and to "**refresh**" the DRAM. Without constant refreshes, DRAM will lose the data written to it as the capacitors leak their charge within a fraction of a second.



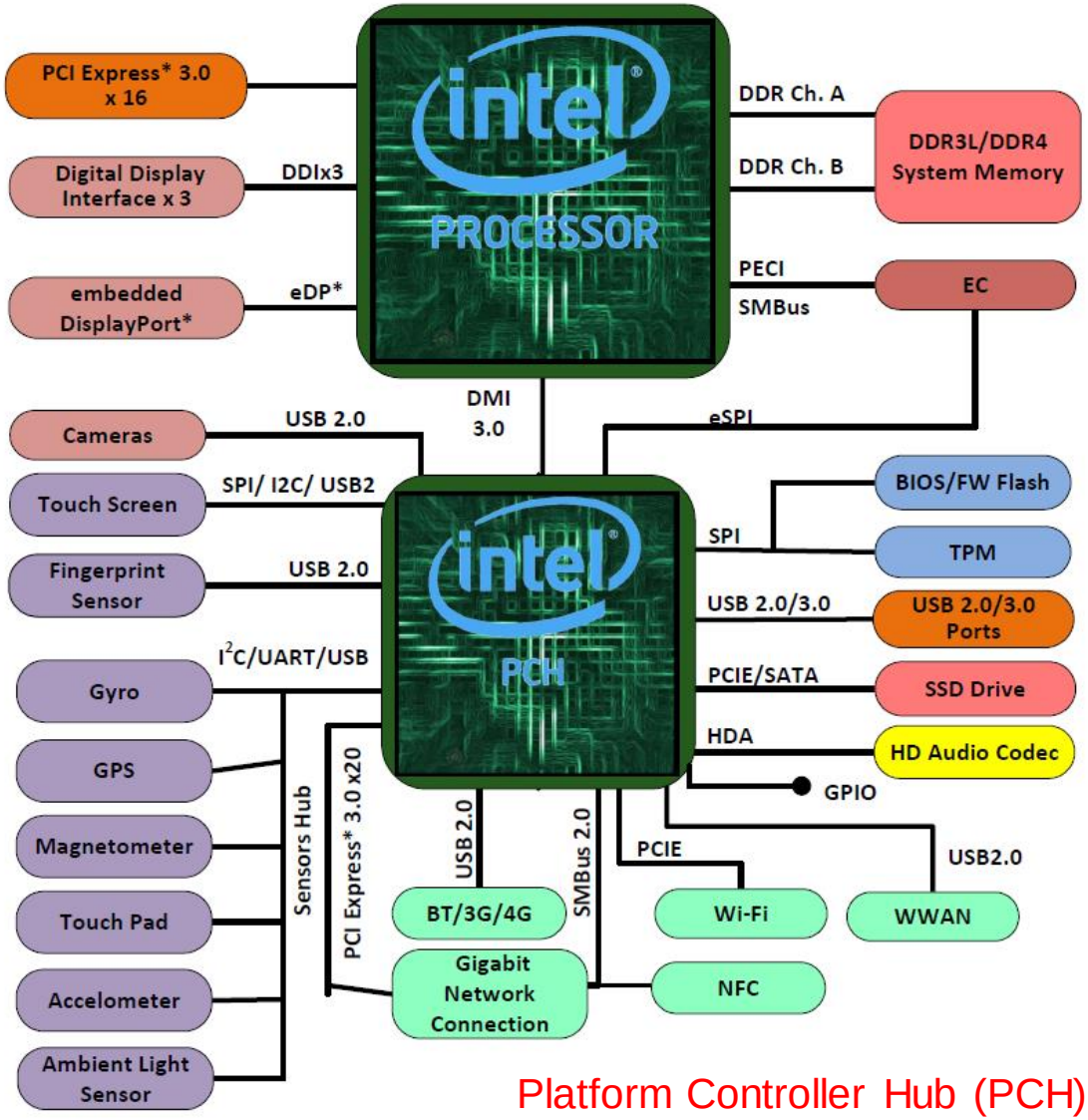
- AMD在2003年把内存控制器内建在CPU里
- Intel在2008年的酷睿i5、i7 系列CPU中也引入了整合内存控制器的方案

内存控制器逐步整合到CPU芯片内

PC主板芯片组背后的故事
2010年后的微型计算机

Intel在推出Nehalem微架构(2008年推出第一款)以后逐步推行单芯片组设计——北桥大部分集成于处理器上，小部分北桥的功能另外集成于剩下的南桥芯片内，是为Intel官方所谓的“PCH单芯片”（Platform Controller Hub）设计。同样，原来集成于北桥的显示核心移步至处理器上。

DMI是Intel开发用于连接主板南北桥的总线，取代了以前的Hub-Link总线。采用点对点的连接方式，时钟频率为100MHz，基于PCI-Express总线。



6th Generation Intel® Core™ Processor Family Datasheet, Volume 1 of 2 http://www.intel.com

Platform Controller Hub (PCH)

消失的北桥：DMI(Direct Media Interface)替代FSB

PC主板芯片组背后的故事
小结：CPU←→Memory、CPU ←→I/O

- PC诞生之初
  - 828x总线控制器、8259终端控制器、……
- 南北桥时代
  - 北主内存、显示；南主I/O
- 消失的北桥　2010～
  - 南桥→PCH(Platform Controller Hub)

- 集成电路规模持续增长（摩尔定律）

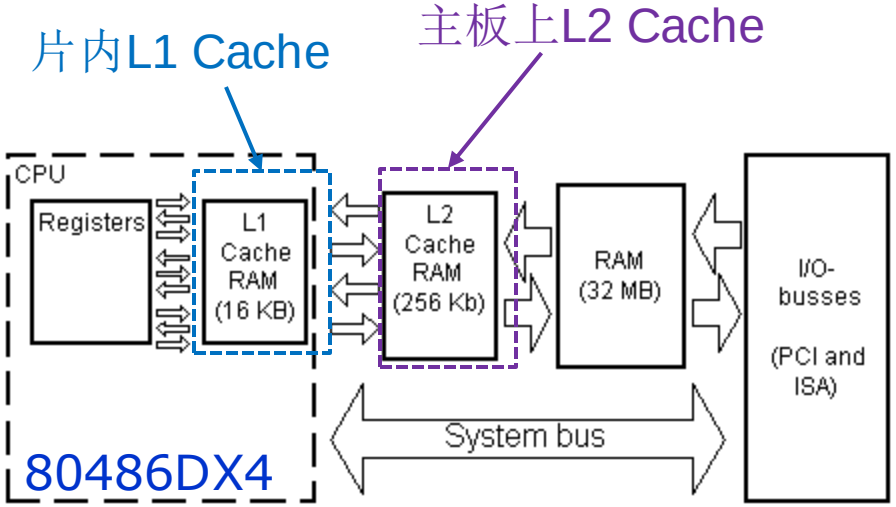
专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

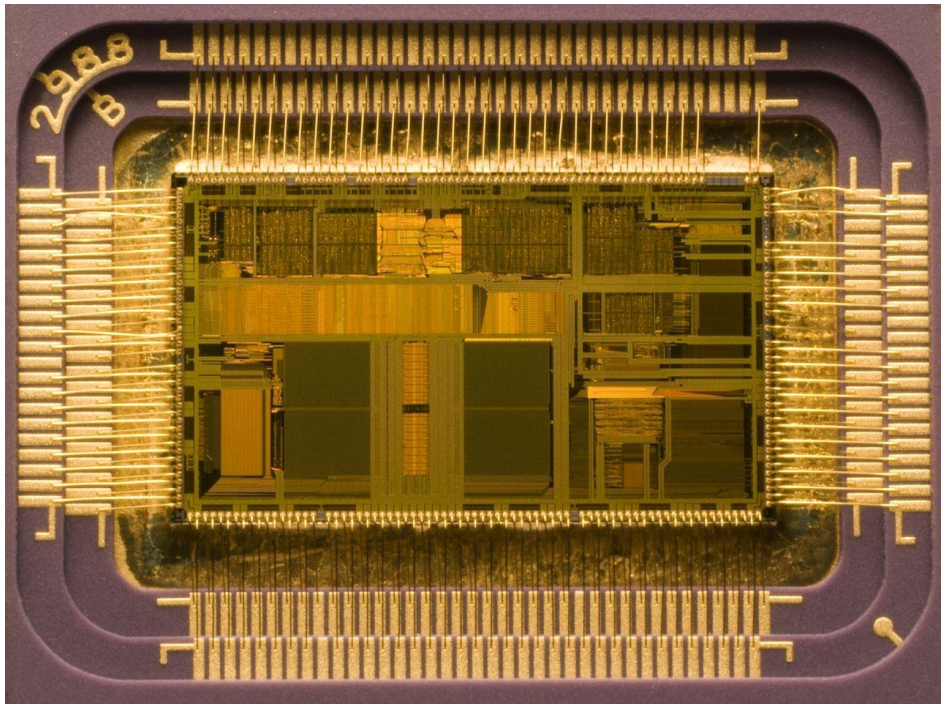
CPU缓存的诞生与发展

80486：开始在片内集成SRAM Cache(L1)

内含8KB的高速缓存，同样是80486处理器的创举。缓存可以用于对频繁访问的指令和数据实现快速的混合存放，使整个芯片的性能得到大幅度提升。缓存概念由此诞生，并一直延续到今天成为影响CPU性能的重要因素。

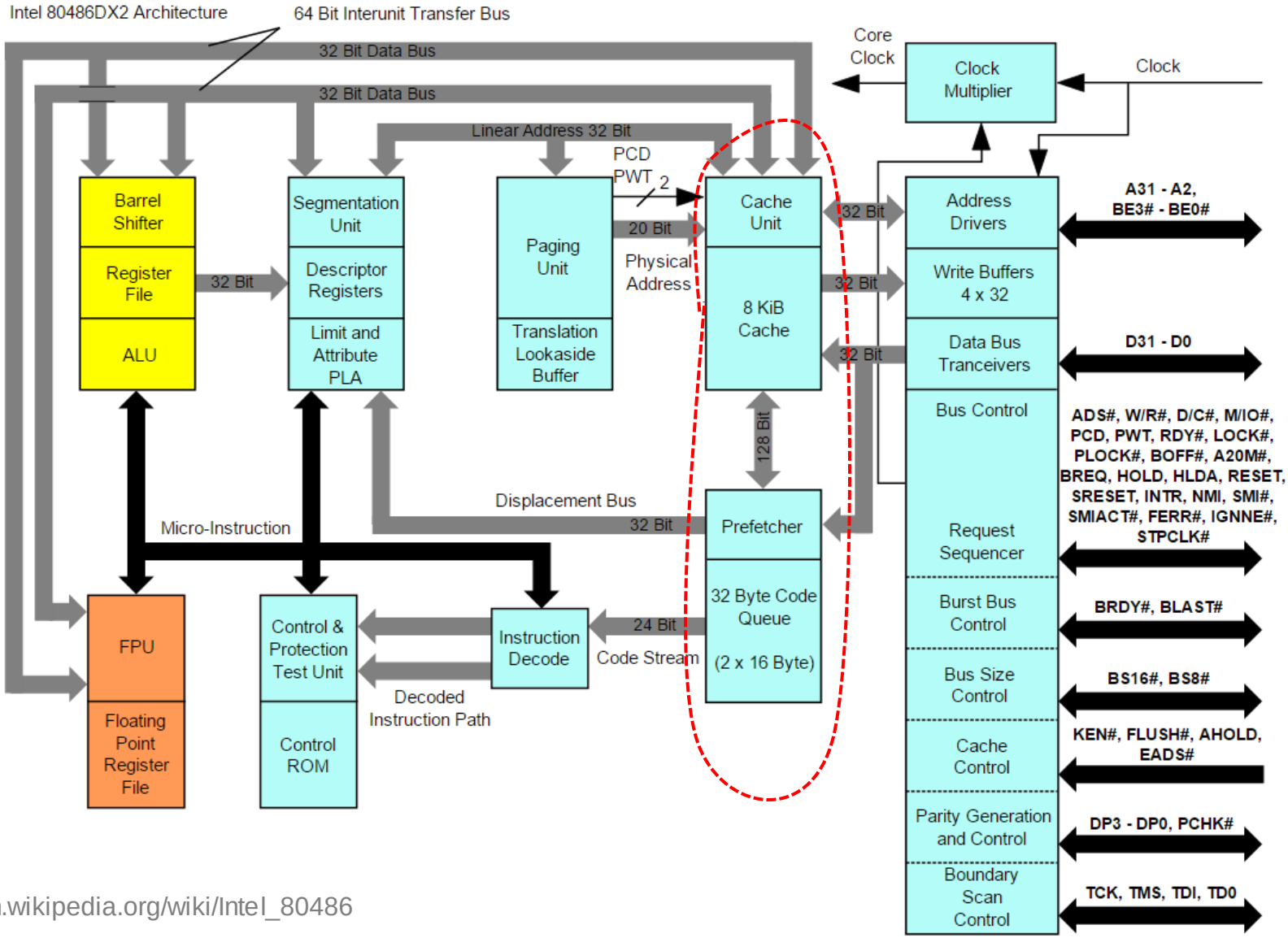


The exposed die of an Intel 80486DX2



https://en.wikipedia.org/wiki/Intel_80486

CPU缓存的诞生与发展
80486片内8kbytes的Cache



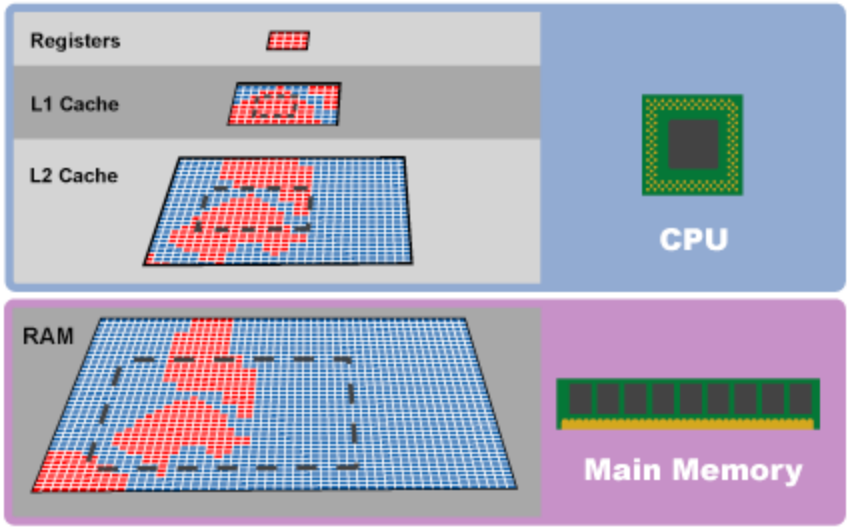
https://en.wikipedia.org/wiki/Intel_80486

CPU缓存的诞生与发展
Pentium Pro之后，集成的二级缓存(L2)

Intel于1996年推出了Pentium Pro（高能奔腾）。该芯片具有两大特色，一是片内封装了与CPU同频运行的256kB或512kB二级缓存；二是支持动态预测执行，可以打乱程序原有指令顺序，这两项改进使得Pentium Pro的性能又有了质的飞跃。

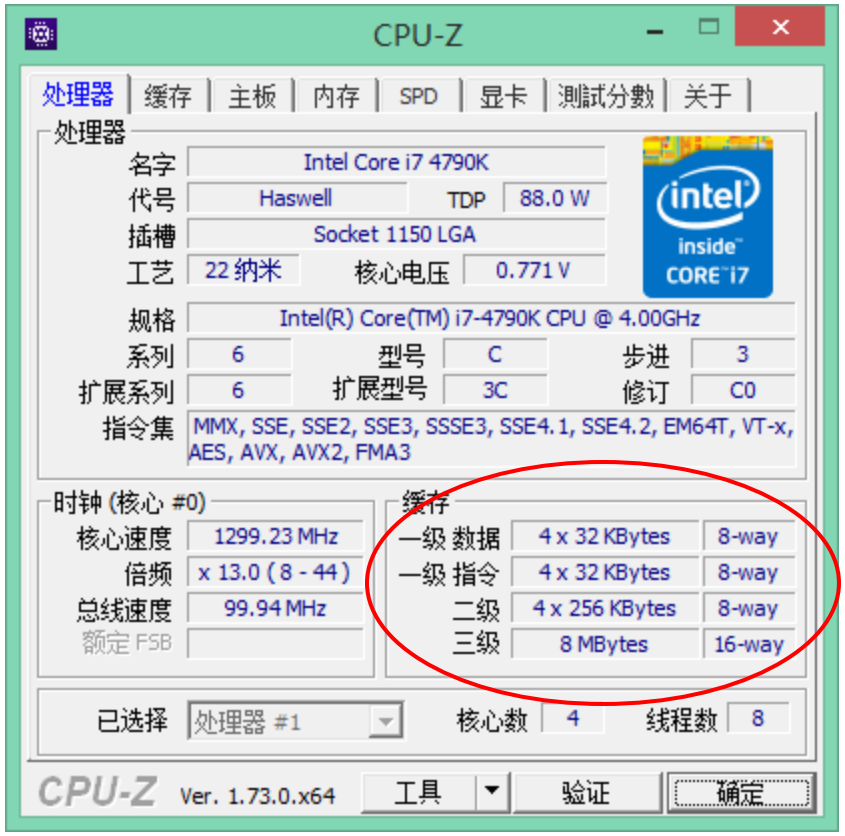
21世纪初各级缓存的容量/速率

Level	Access Time	Typical Size	Technology	Managed By
Registers	1-3 ns	?1 KB	Custom CMOS	Compiler
Level 1 Cache (on-chip)	2-8 ns	8 KB - 128 KB	SRAM	Hardware
Level 2 Cache (off-chip)	5-12 ns	0.5 MB - 8 MB	SRAM	Hardware
Main Memory	10-60 ns	64 MB - 1 GB	DRAM	Operating System
Hard Disk	3,000,000 - 10,000,000 ns	20GB - 100 GB	Magnetic	Operating



http://arstechnica.com/gadgets/2002/07/caching/

CPU缓存的诞生与发展
Intel i7：片内集成L3 Cache

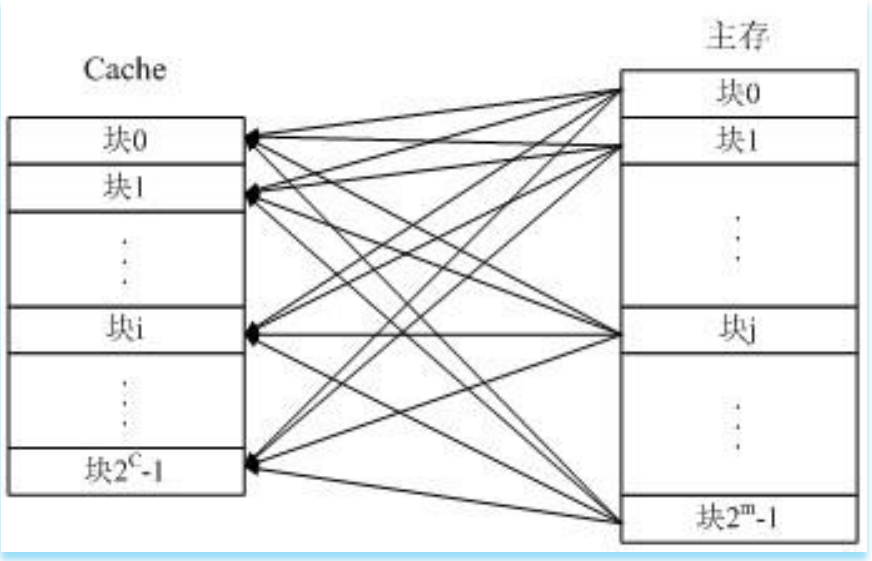


程序的局部性原理（代码和数据访问的特性）

- Spatial locality：将最近被访问的信息装入Cache
- Temporal locality：将最近被访问信息的临近信息装入Cache

Cache原理：Memory到Cache的映射

Cache与主存都分成块(常成为Cache line)，每块由多个字节组成，大小相等。在一个时间段内，Cache的某块中放着主存某块的全部信息，即Cache的某一块是主存某块的副本(或叫映像)。

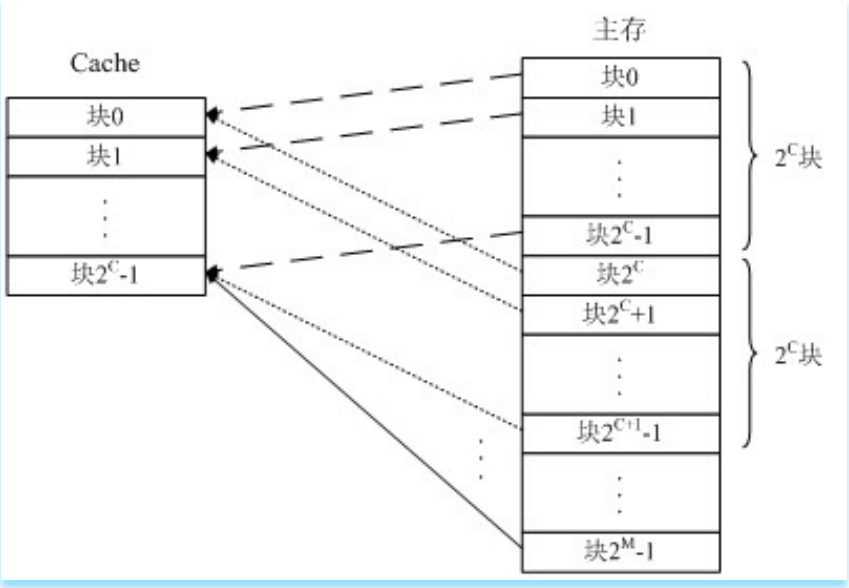


全相联映射方式（Fully Associative）：主存中任意一个块都可以映射到Cache中任意一个块。设Cache共有2^C块，主存共有2^M块，当主存的某一块j需调进Cache中时，它可以存入Cache的块0、块1、…、块i、…或块2^C-1的任意一块上。

Cache原理：Memory到Cache的映射

直接相联映射(Direct-Mapped)方式是指主存的某块j只能映射到满足如下特定关系的Cache块i中：**$i=j \bmod 2^C$** 。如主存的第0、2^C、2^{C+1}、…块只能映射到Cache的第0块，主存的第1、2^C+1、2^{C+1}+1、…块只能映射到Cache的第1块，……，主存的第2^C-1、2^{C+1}-1、…2^M-1块只能映射到Cache的第2^C-1块。

全相联映射方式命中率比较高，Cache存储空间利用率高。但相联存储器庞大，电路复杂，访问相关存储器时，速度低，只适合于小容量Cache，应用少。直接相联映射方式硬件电路简单，但命中率低， 当主存的某块需调入Cache时，如果对应的Cache特定块已被占用，而Cache中的其它块即使空闲，也需要替换整组Cache。



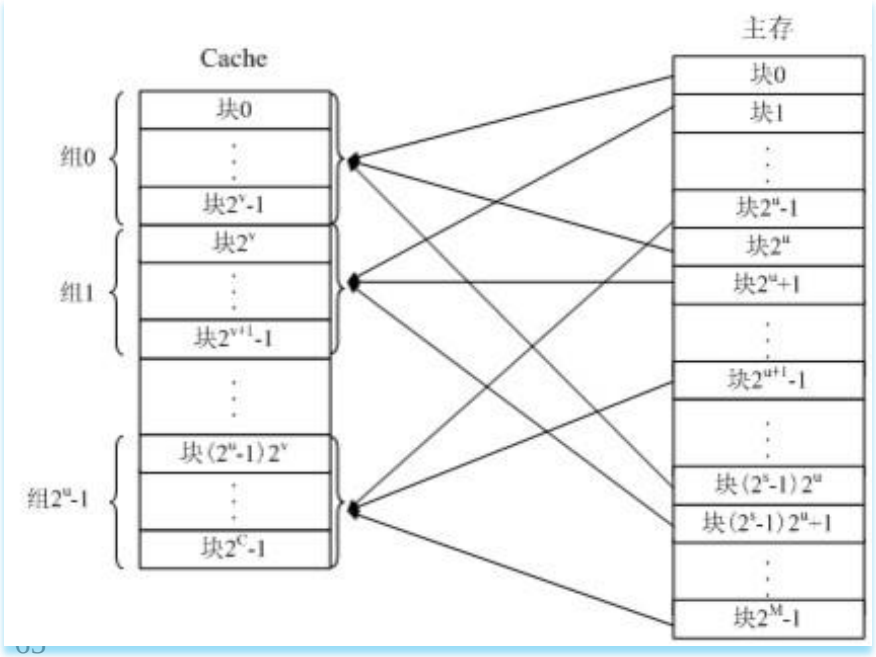
Cache原理：Memory到Cache的映射

组相联映射(Set-Associative)方式：将主存分为$2^s \times 2^u$块；Cache分成2^u组，每组包含2^v块。主存的块与Cache的组之间采用直接相联映射，而与组内的各块则采用全相联映射。也就是说，主存的某块只能映射到Cache的特定组中的任意一块。主存的某块j与Cache的组k之间满足如下关系：　　$k = j \bmod 2^u$

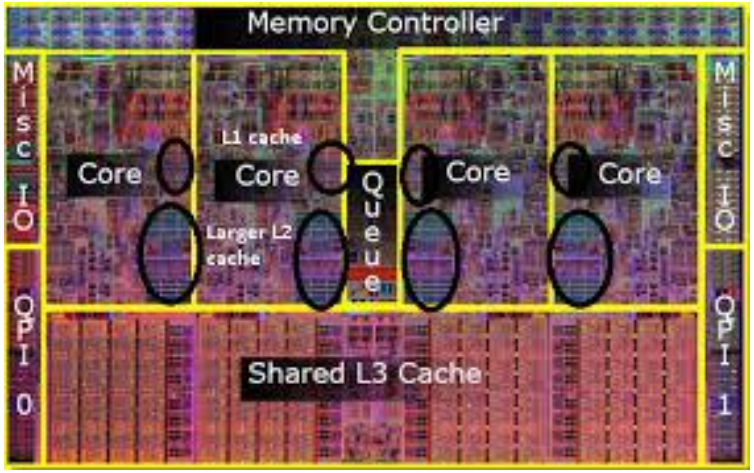
例8-entry cache采用2-Way Set-Associative方式，将8个Cache分成了2组，每组有4个Cache Line。

最常见的方式

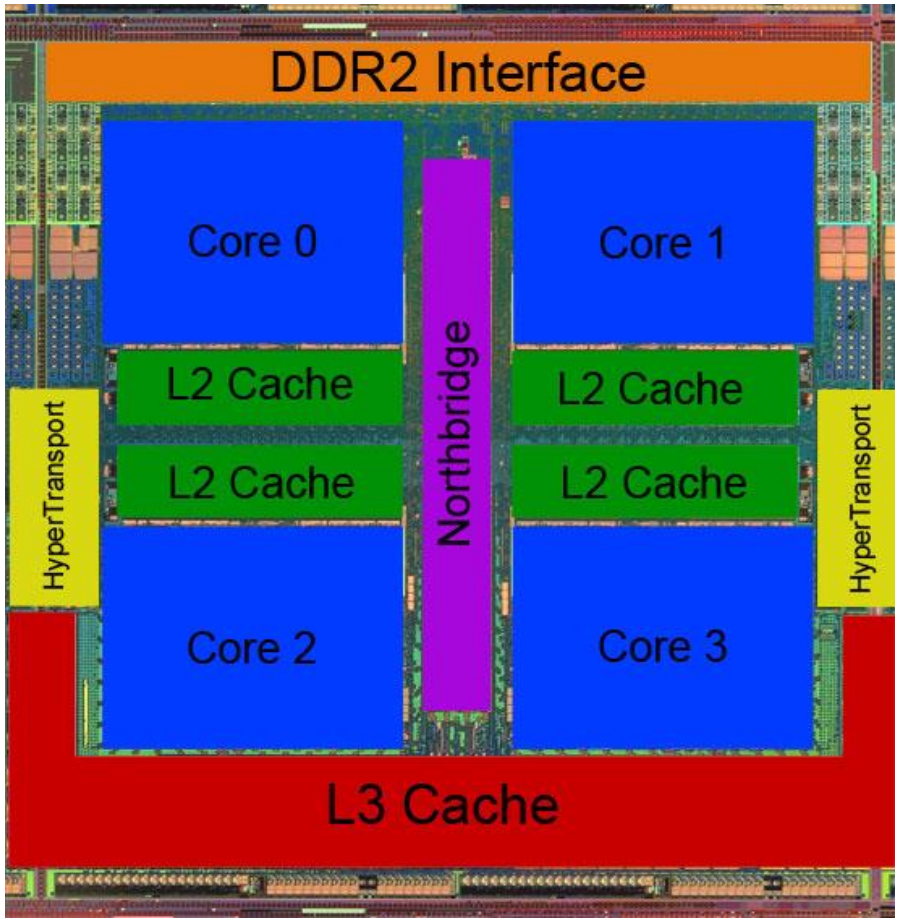
集合了全相联映射方式和直接相联映射方式的特点。块的冲突概率比较低，块的利用率大幅度提高，块失效率明显降低。缺点：实现难度和造价要比直接映射方式高。



CPU缓存的诞生与发展
i7之后，集成的三级缓存(L3)



Intel i7 (2008)



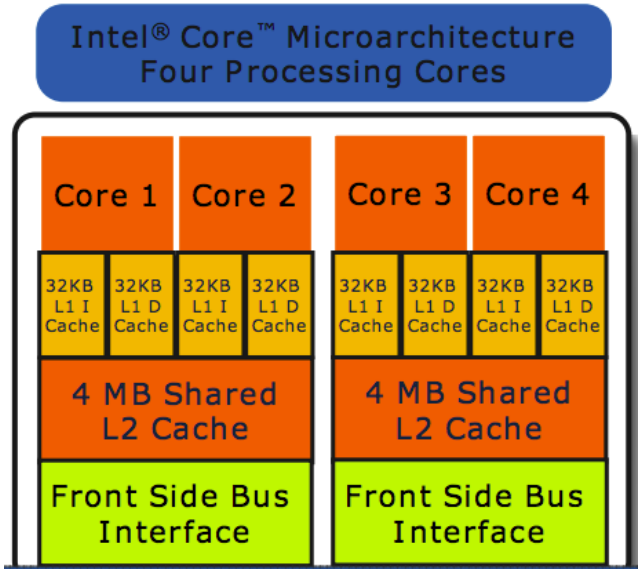
AMD Phenom II (2008)

http://www.anandtech.com/show/2702/2

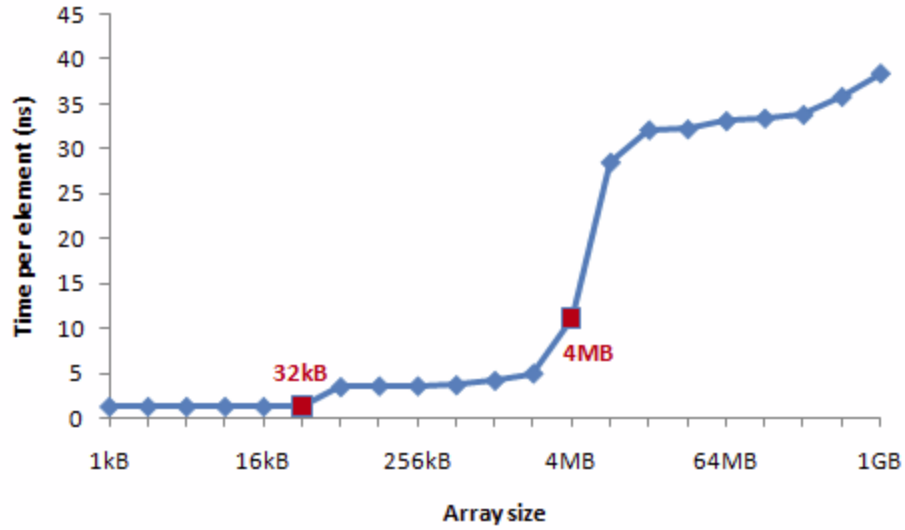
L1和L2容量对代码执行效率的影响

4核CPU：a 32kB L1 data cache, a 32kB L1 instruction cache, and a 4MB L2 data cache. The L1 caches are per-core, and the L2 caches are shared between pairs of cores.

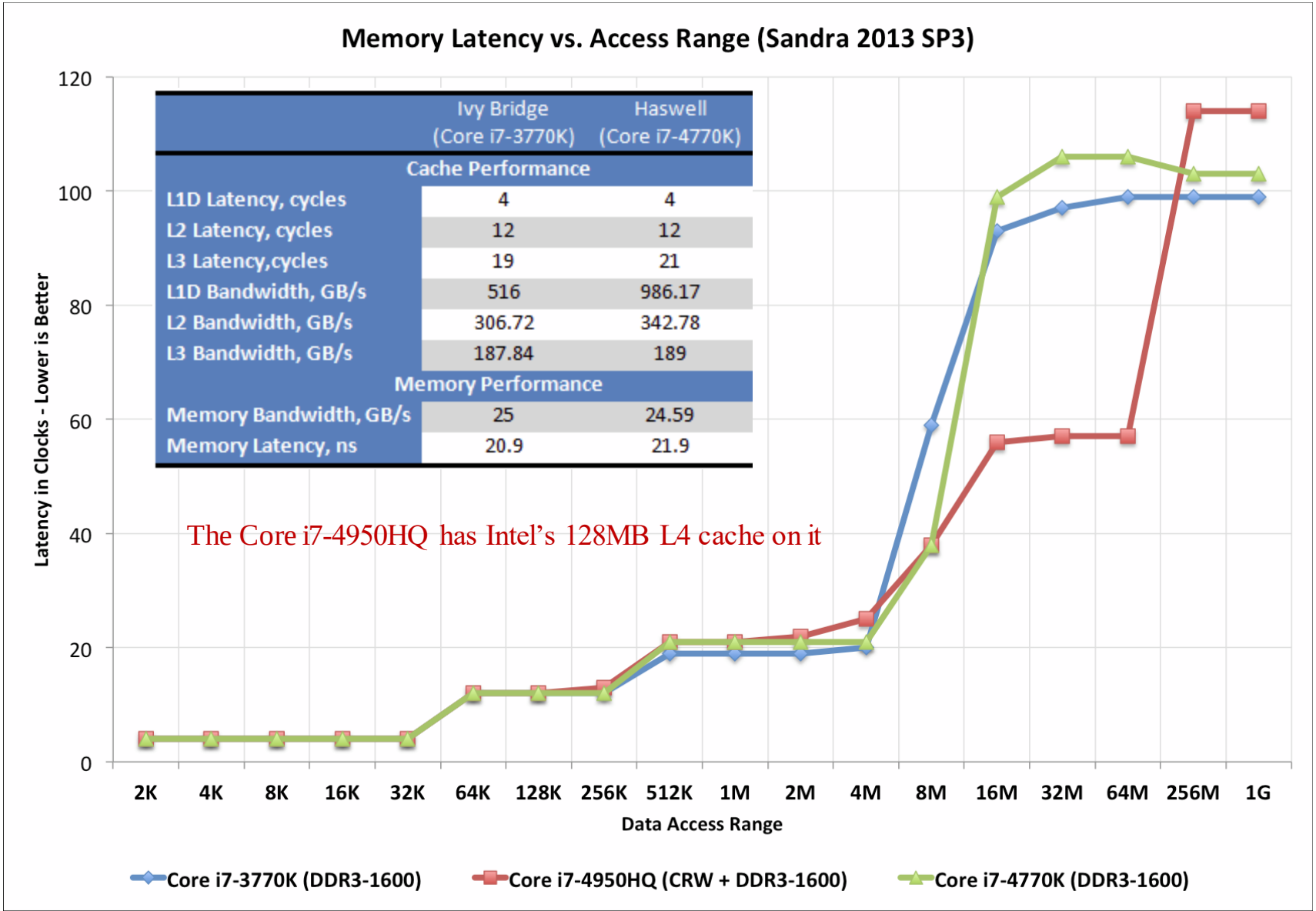
```
*--- Data Cache         0, Level 1,   32 KB, Assoc   8, LineSize   64
*--- Instruction Cache  0, Level 1,   32 KB, Assoc   8, LineSize   64
-*-- Data Cache         1, Level 1,   32 KB, Assoc   8, LineSize   64
-*-- Instruction Cache  1, Level 1,   32 KB, Assoc   8, LineSize   64
**-- Unified Cache      0, Level 2,   4 MB, Assoc  16, LineSize   64
--*- Data Cache         2, Level 1,   32 KB, Assoc   8, LineSize   64
--*- Instruction Cache  2, Level 1,   32 KB, Assoc   8, LineSize   64
---* Data Cache         3, Level 1,   32 KB, Assoc   8, LineSize   64
---* Instruction Cache  3, Level 1,   32 KB, Assoc   8, LineSize   64
--** Unified Cache      1, Level 2,   4 MB, Assoc  16, LineSize   64
```




```
int steps = 64 * 1024 * 1024;
int lengthMod = arr.Length - 1;
for (int i = 0; i < steps; i++)
{
    arr[(i * 16) & lengthMod]++;
}
```



数据块大小 vs. 访问延迟



	Ivy Bridge (Core i7-3770K)	Haswell (Core i7-4770K)
Cache Performance		
L1D Latency, cycles	4	4
L2 Latency, cycles	12	12
L3 Latency,cycles	19	21
L1D Bandwidth, GB/s	516	986.17
L2 Bandwidth, GB/s	306.72	342.78
L3 Bandwidth, GB/s	187.84	189
Memory Performance		
Memory Bandwidth, GB/s	25	24.59
Memory Latency, ns	20.9	21.9

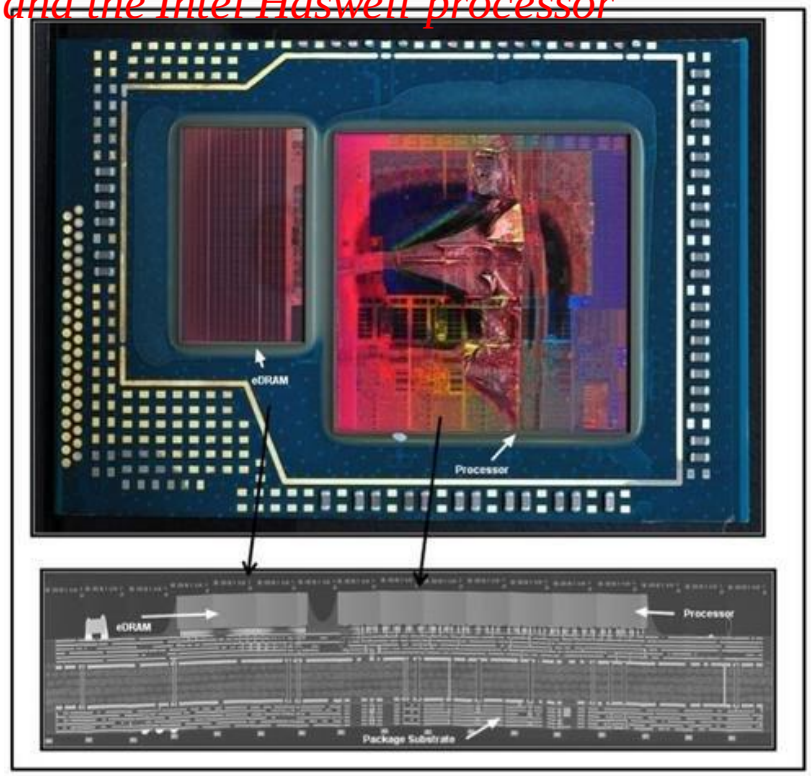
The Core i7-4950HQ has Intel's 128MB L4 cache on it

CPU缓存的诞生与发展
未来？集成的四级缓存(L4)

eDRAM stands for "embedded DRAM", a capacitor-based dynamic random-access memory integrated on the same die or module as an ASIC or processor. eDRAM's cost-per-bit is higher when compared to equivalent standalone DRAM chips used as external memory, but the performance advantages of placing eDRAM onto the same chip as the processor outweigh the cost disadvantages in many applications.

Product name	Amount of eDRAM
Intel Haswell, Iris Pro Graphics 5200 (GT3e)	128 MB
Intel Broadwell, Iris Pro Graphics 6200 (GT3e)	128 MB
Intel Skylake, Iris Graphics 540 and 550 (GT3e)	64 MB
Intel Skylake, Iris Pro Graphics 580 (GT4e)	64 or 128 MB
PlayStation 2	4 MB
Xbox 360	10 MB

Intel GT3e GPU containing eDRAM and the Intel Haswell processor



eDRAM (embedded DRAM)

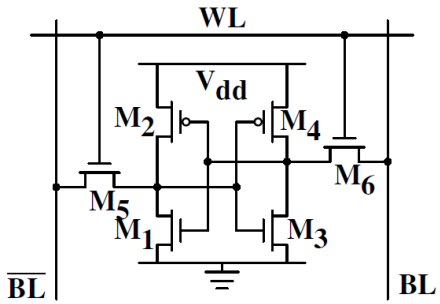


Figure 2.4: 6-T Static RAM

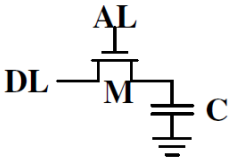
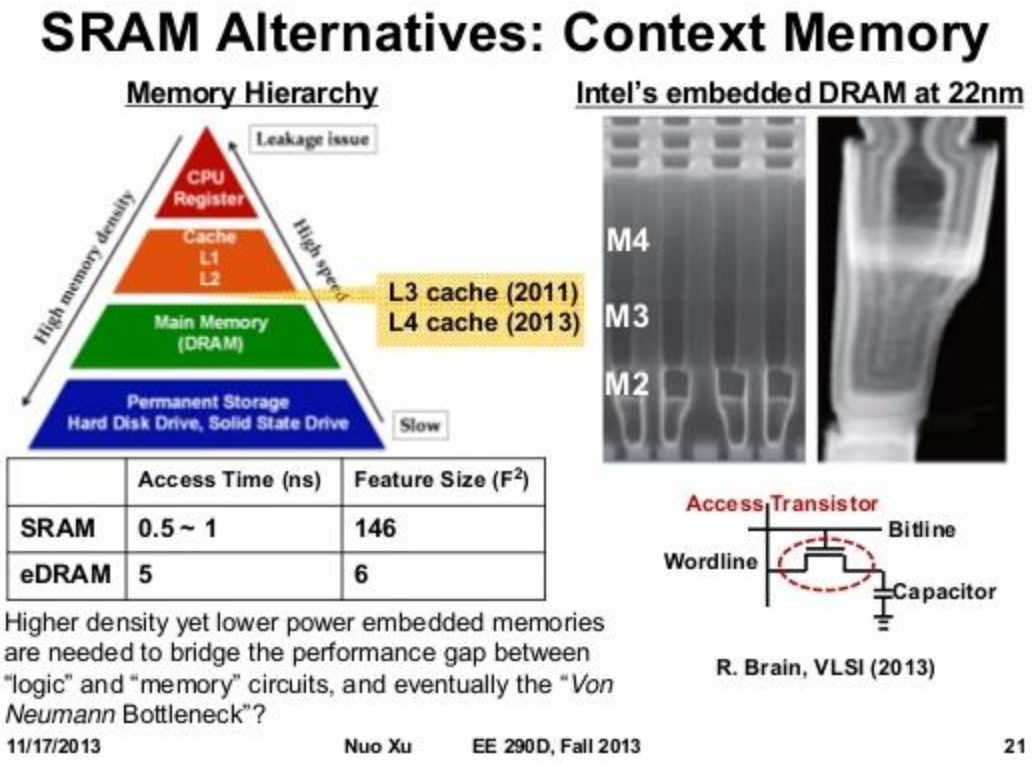


Figure 2.5: 1-T Dynamic RAM



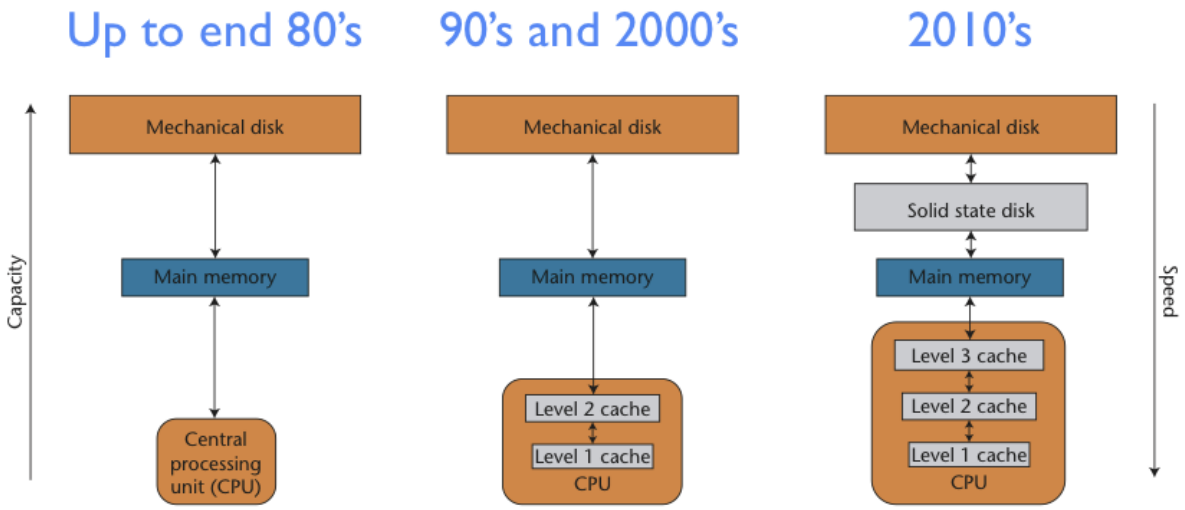
	Access Time (ns)	Feature Size (F^2)
SRAM	0.5 ~ 1	146
eDRAM	5	6

EDRAM cell area at the 22-nm process technology, reduced to 0,029 square meters. um. SRAM cell area in the same process technology, is 0,092 square meters. - is three times more. That is eDRAM is much denser and energy-efficient usual cache.

Cache的经典文章
What Every Programmer Should Know About Memory
Ulrich Drepper, Red Hat, Inc. November 21, 2007
https://people.freebsd.org/~lstewart/articles/cpumemory.pdf

小结：

- 片内：L1→L2→L3→… → L4
- SRAM、eDRAM、DRAM



Cache的经典文章
What Every Programmer Should Know About Memory
Ulrich Drepper, Red Hat, Inc. November 21, 2007
https://people.freebsd.org/~lstewart/articles/cpumemory.pdf

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

总线的发展

PCI Express (Peripheral Component Interconnect Express)

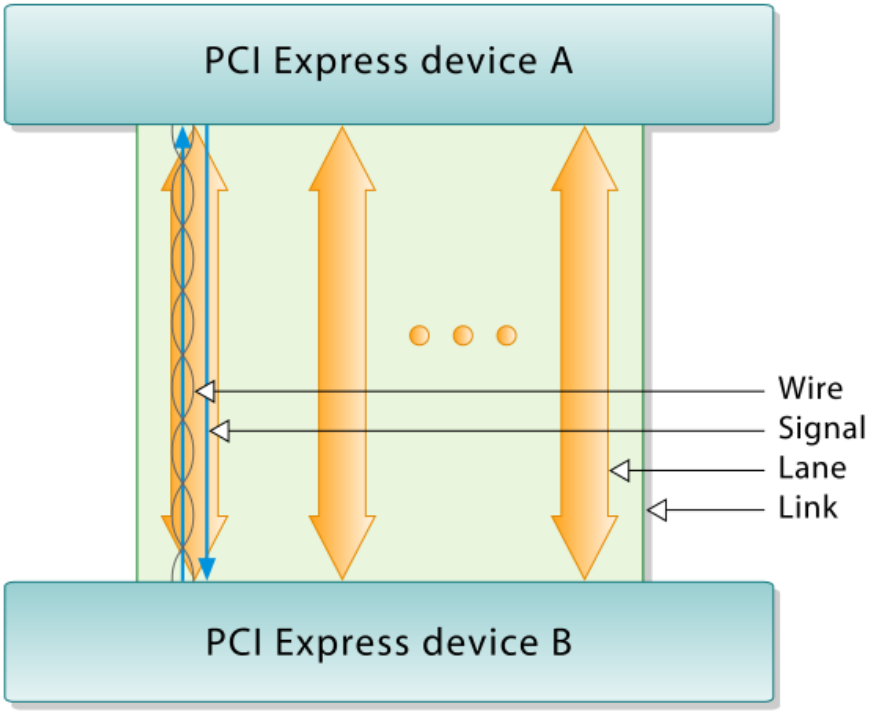
	ISA	**PCI**	**AGP (1X-8X)**	**PCI-X 1.0**	**PCI-X 2.0 (DDR/QDR)**	**PCI-E 1(X1-X16)**	**PCI-E 2(X1-X32)**	**PCI-E 3(X1-X32)**	**PCI-E 4(X1-X32)**
位宽	8/16	32/64	32	32/64	64				
总线时钟	8	33/66	66	66/100/133	133				
最高带宽		533MB	266-2132MB	1066MB	2132/4264MB	per-lane 2.5GT/s	per-lane 5GT/s	per-lane 8GT/s	per-lane 16GT/s
商用时间	Early 80's	Early 90's	Mid/late 90's	1999		2004	2007	2009	2011

GT/s (gigatransfers per second)

总线的发展
PCIE是串行总线

A lane is composed of **two differential signaling pairs**, with one pair for receiving data and the other for transmitting. Thus, each lane is composed of four wires or signal traces. Conceptually, each lane is used as a full-duplex byte stream, transporting data packets in eight-bit "byte" format simultaneously in both directions between endpoints of a link.

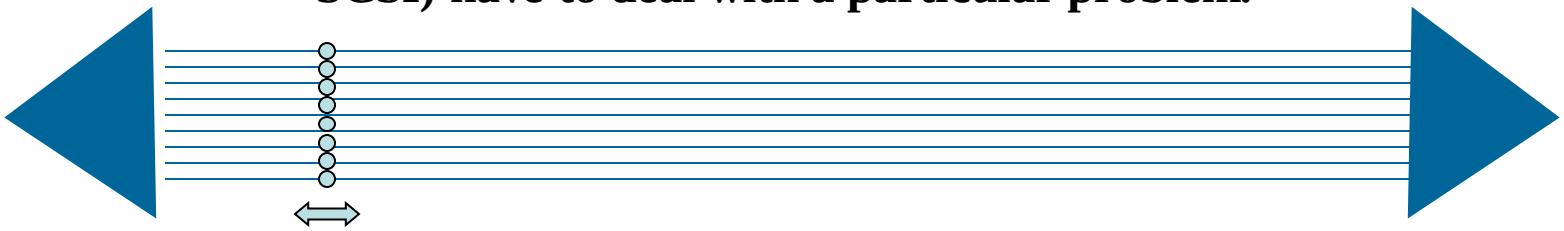
Physical PCI Express links may contain from one to 32 lanes, more precisely 1, 2, 4, 8, 12, 16 or 32 lanes. Lane counts are written with an "×" prefix (for example, "×8" represents an eight-lane card or slot), with ×16 being the largest size in common use.



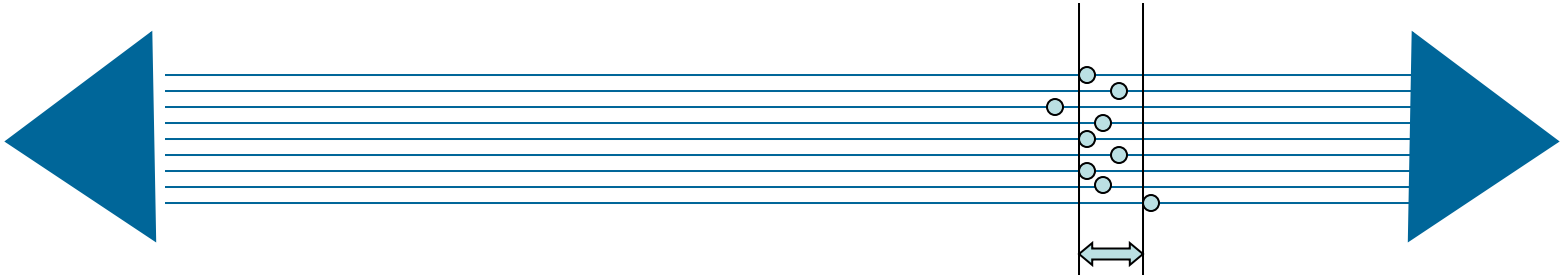
https://en.wikipedia.org/wiki/PCI_Expres

总线的发展
并行总线的瓶颈

Computer Buses, both internal (e.g. PCI) and external (e.g. SCSI) have to deal with a particular problem.



They have to keep the data bits across the connectors in line so that they all arrive together, within a very tight time period

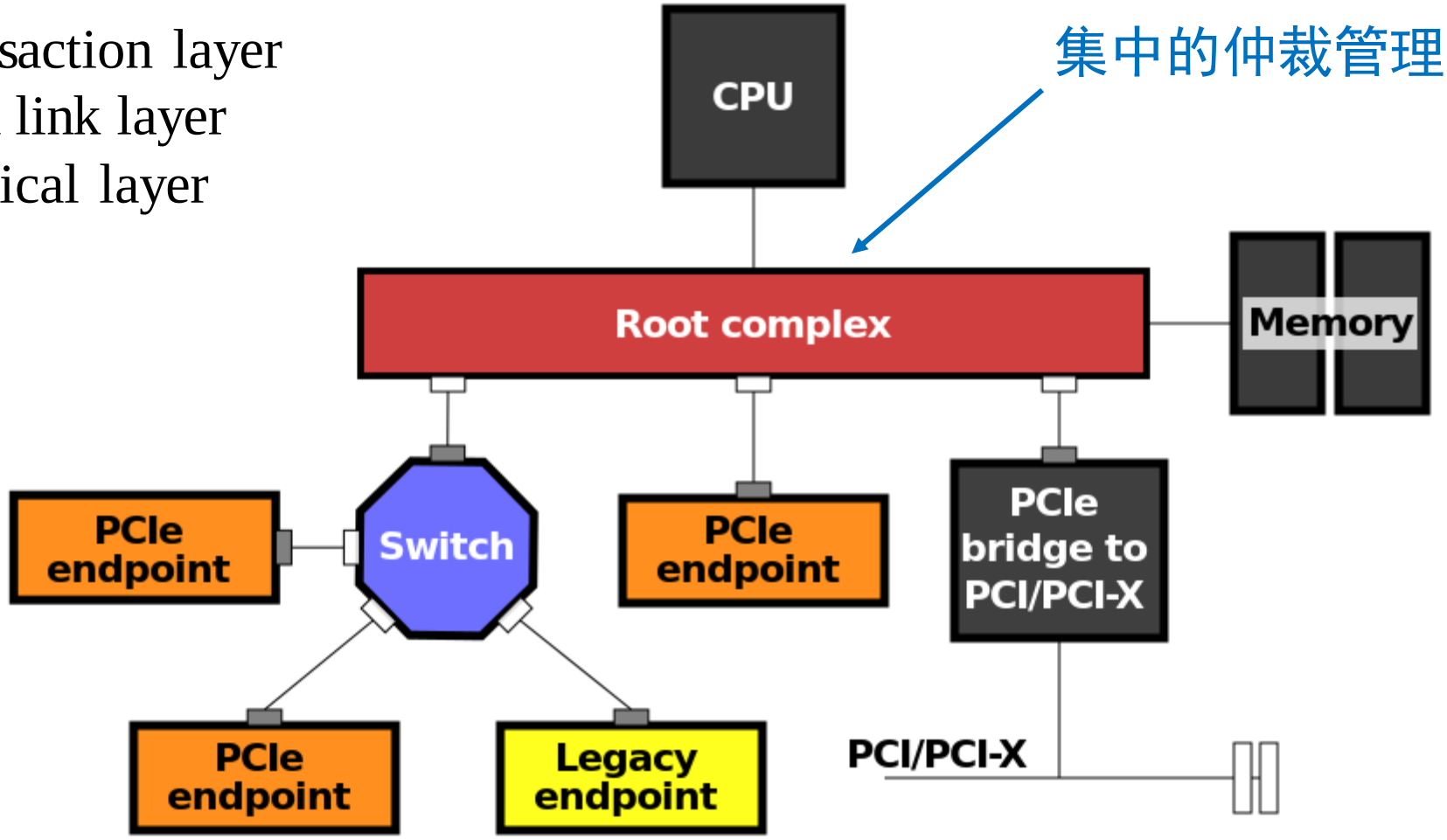


The further the information moves down a parallel bus the more the minute differences between individual connections come into play and the original data starts to scatter (skew) becoming corrupt and unusable.

This skew effect limits the distance over which high speed, parallel bus connections can be used

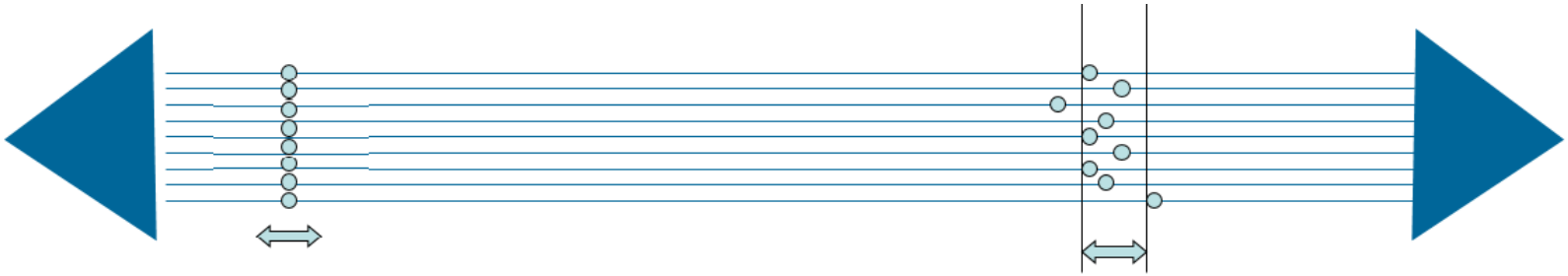
总线的发展
PCIE总线互联类似计算机局域网的结构

Transaction layer
Data link layer
Physical layer



小结：总线的发展

- 串行化是提高传输时钟的必然要求
  - PCI-X 133MHz
  - PCIE 2.5GHz
  - PCIE4 16GHz



- 其他类似串行总线
  - 南北桥：DMI(Direct Media Interface)
  - 硬盘：SATA

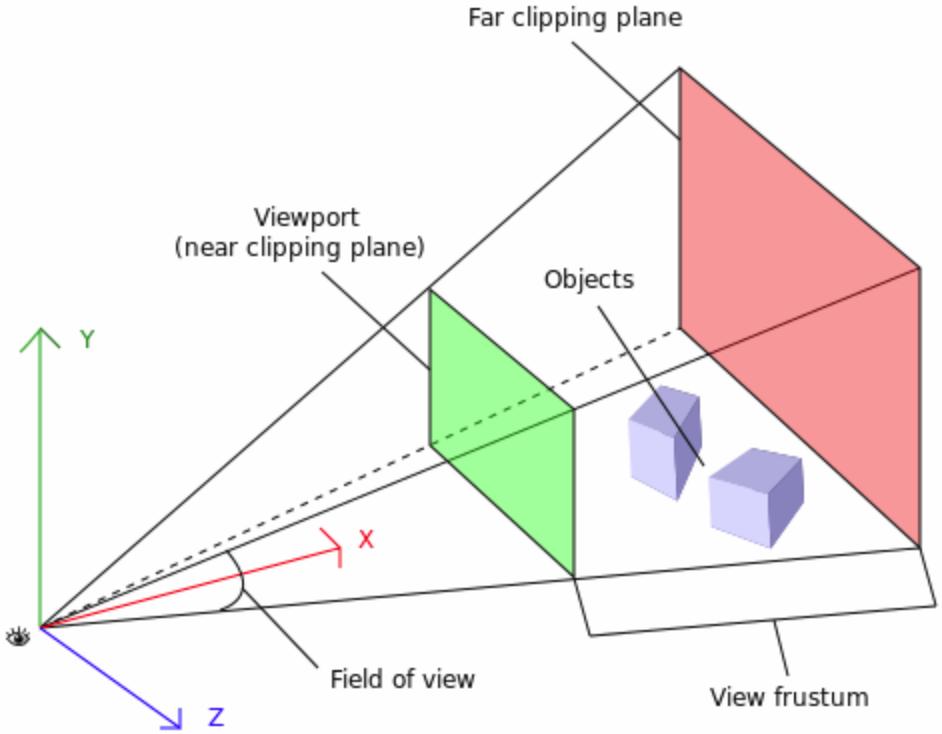
专题：多媒体计算→异构计算

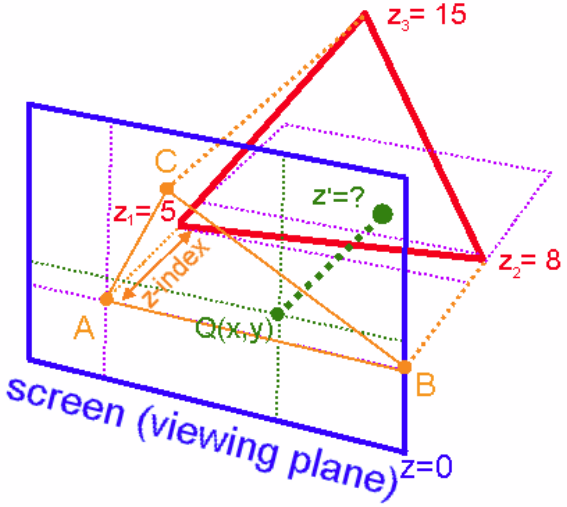
- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

2D/3D图形计算的需求
顶点坐标变换 Vertex Processing

从3D模型到屏幕显示，模型中的顶点(Vertex)需要经过不同坐标系的变换。

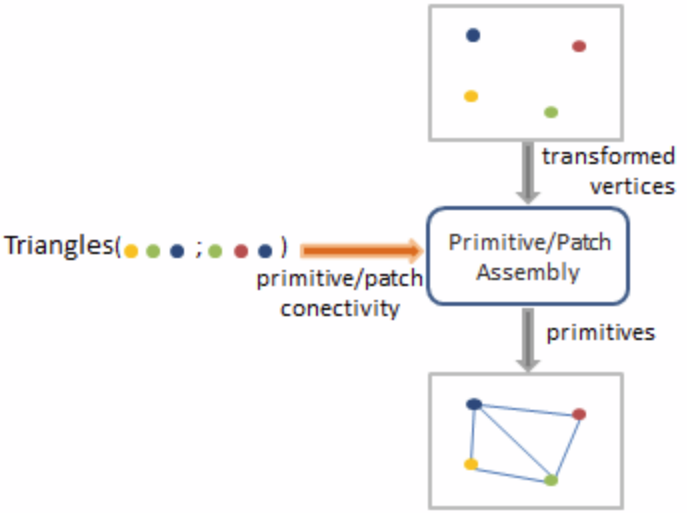
- Object space，模型坐标空间
- World space，世界坐标空间
- Eye space，观察坐标空间
- Clip and Project space，屏幕坐标空间

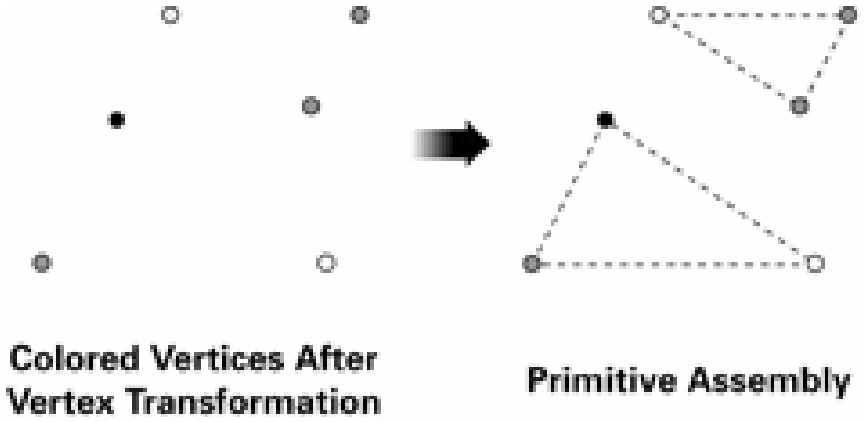




http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter04.html

2D/3D图形计算的需求
图元拼装 Primitive Assembly

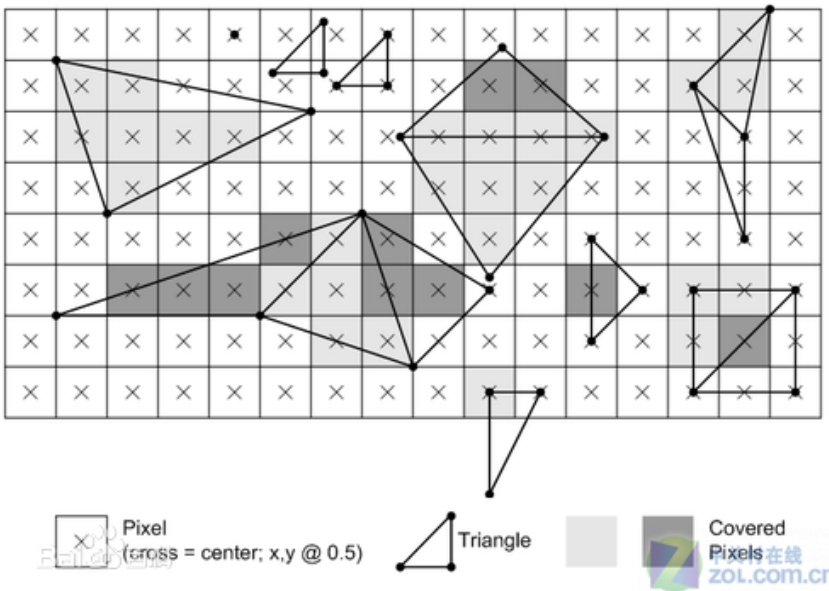


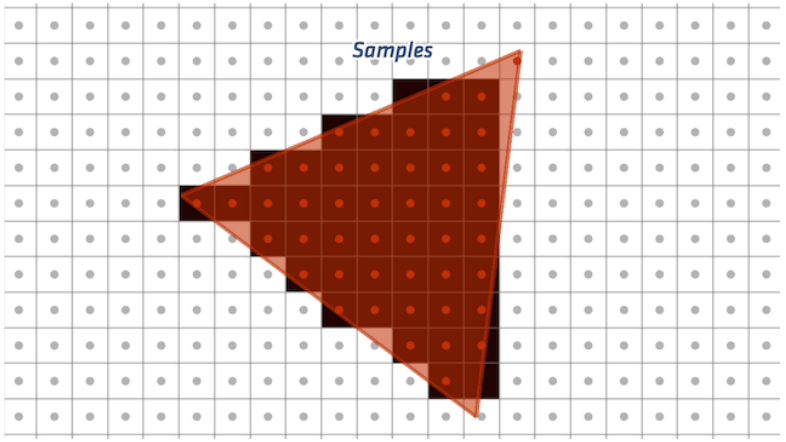


2D/3D图形计算的需求
光栅化(Rasterization)

Rasterisation (or rasterization) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format.

光栅化：把景物模型的数学描述（显示列表）及其色彩信息转换至计算机显示器上的像素。

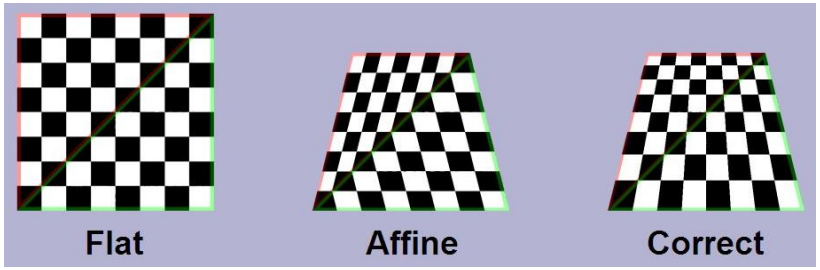


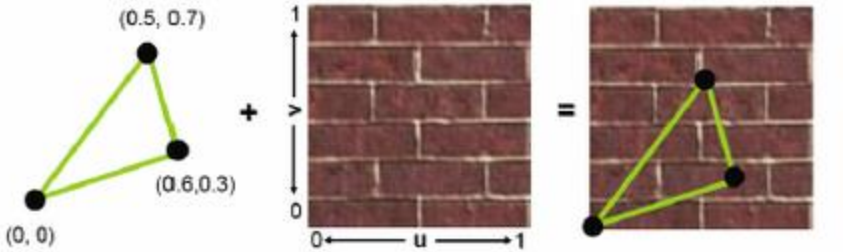


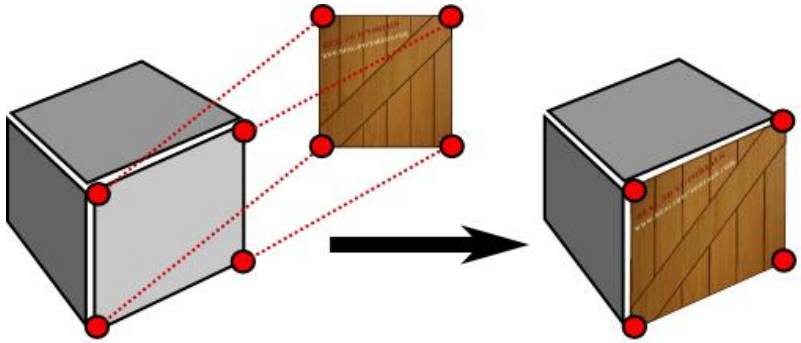
2D/3D图形计算的需求
纹理映射(Texture Mapping)

Texture mapping is a method for adding detail, surface texture (a bitmap or raster image), or color to a computer-generated graphic or 3D model.

纹理映射，又称纹理贴图，是将纹理空间中的纹理像素映射到屏幕空间中的像素的过程。简单来说，就是把一幅图像贴到三维物体的表面上来增强真实感，可以和光照计算、图像混合等技术结合起来形成许多非常漂亮的效果。

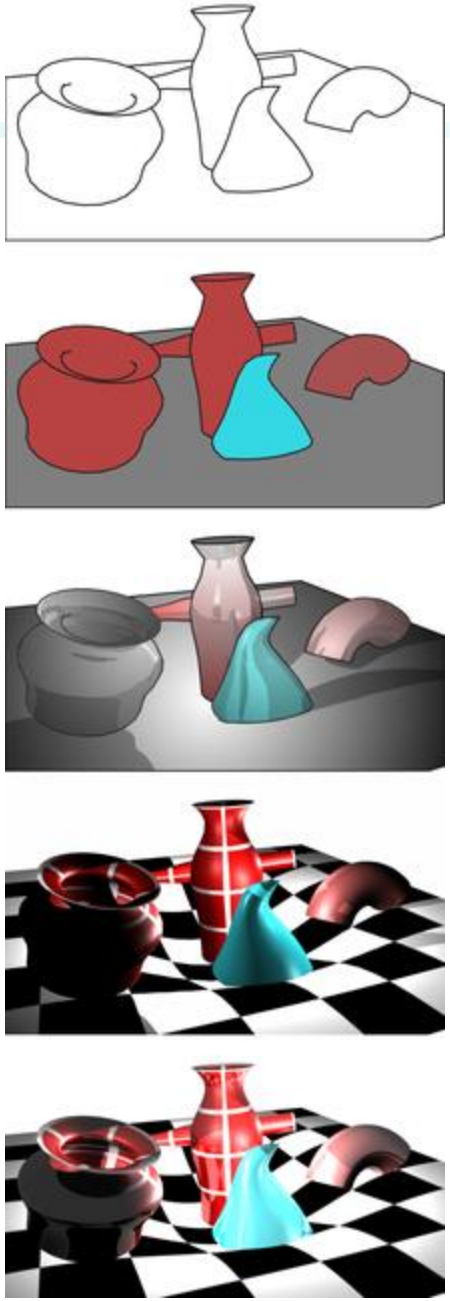






2D/3D图形计算的需求
渲染(Rendering)

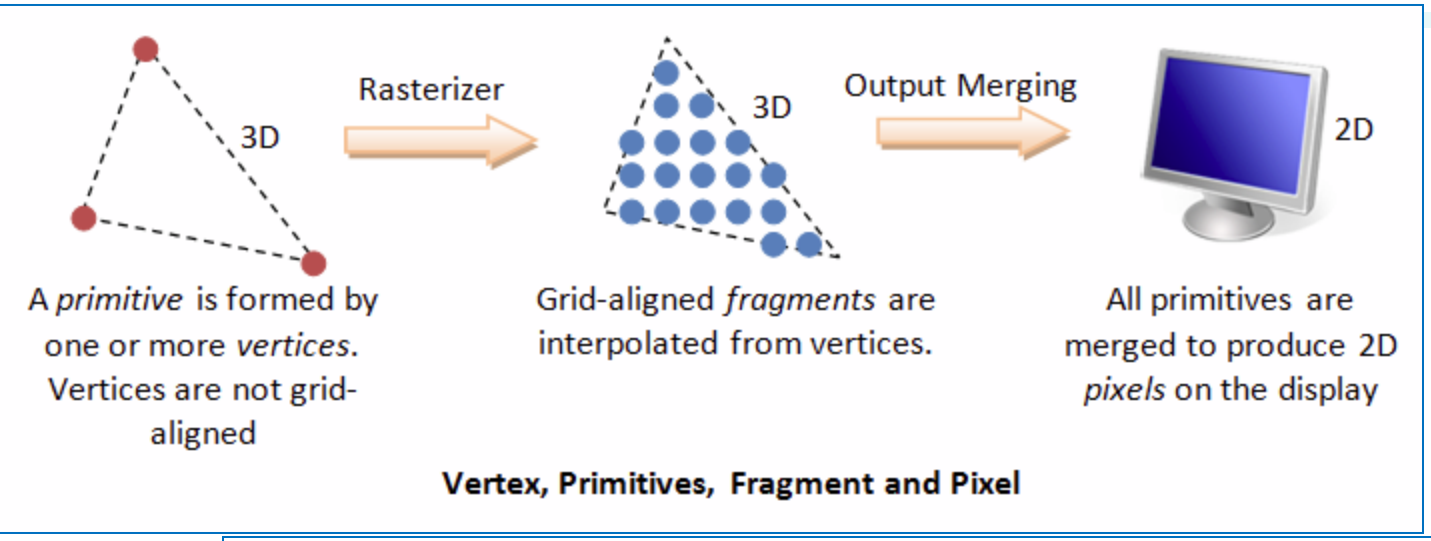
Rendering is the process of generating an image from a 2D or 3D model (or models in what collectively could be called a scene file), by means of computer programs. Also, the results of such a model can be called a rendering. A scene file contains objects in a strictly defined language or data structure; it would contain geometry, viewpoint, texture, lighting, and shading information as a description of the virtual scene.



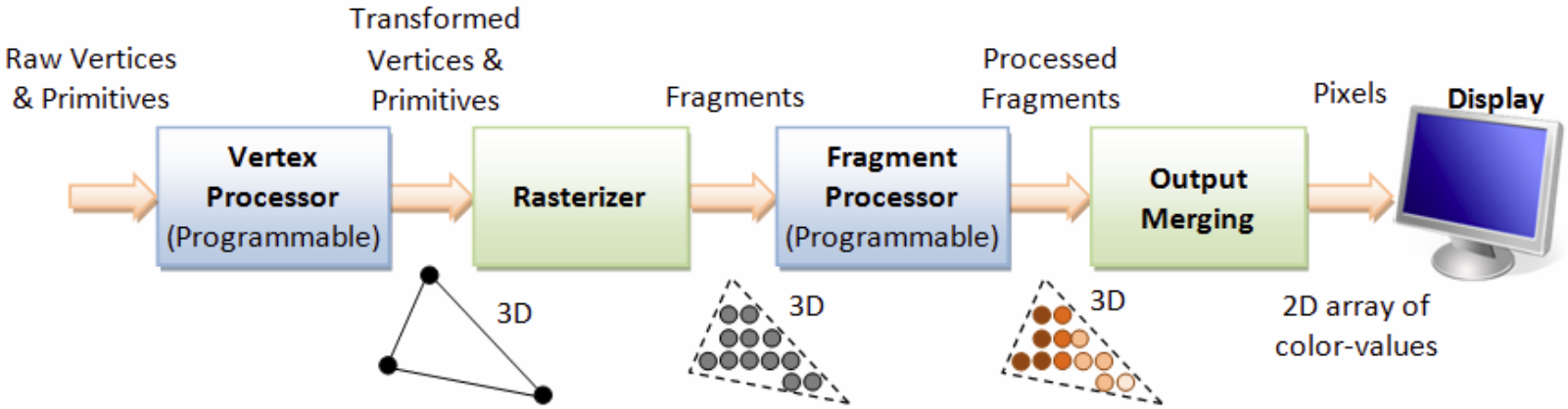
专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

计算机如何创建图像？
图元(primitive)→片段(fragment)→像素(pixel)



Vertex, Primitives, Fragment and Pixel

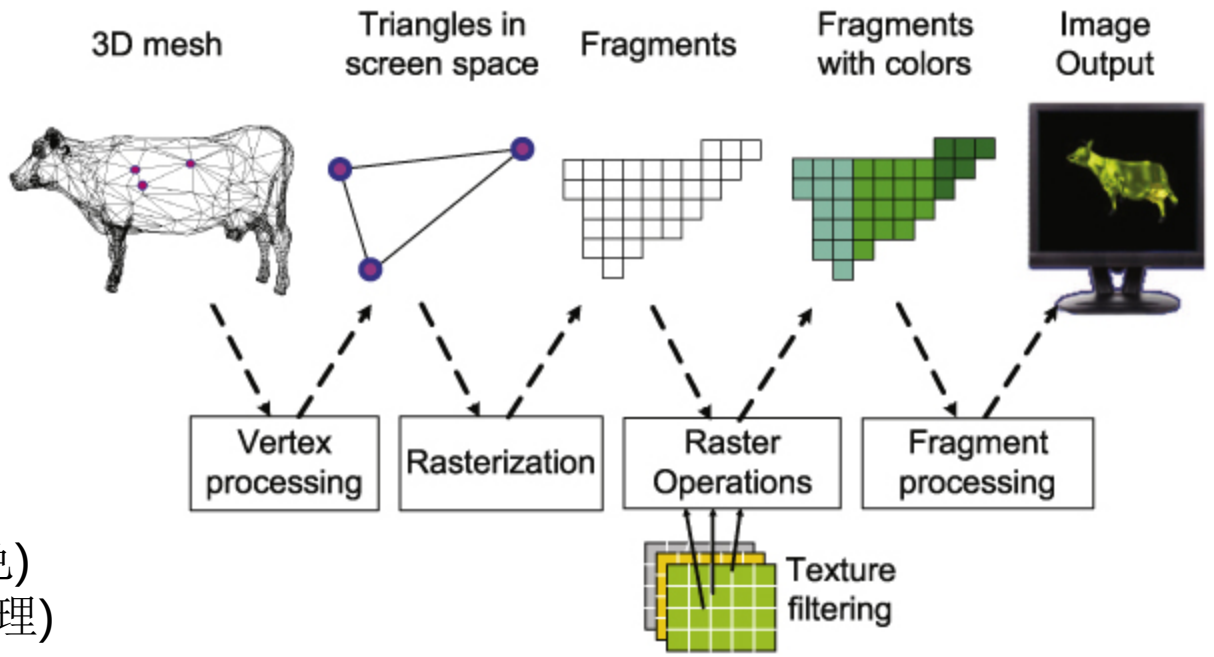


3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

3D Graphics with OpenGL
http://www.ntu.edu.sg/home/ehchua/programming/opengl/cg_basicstheory.html

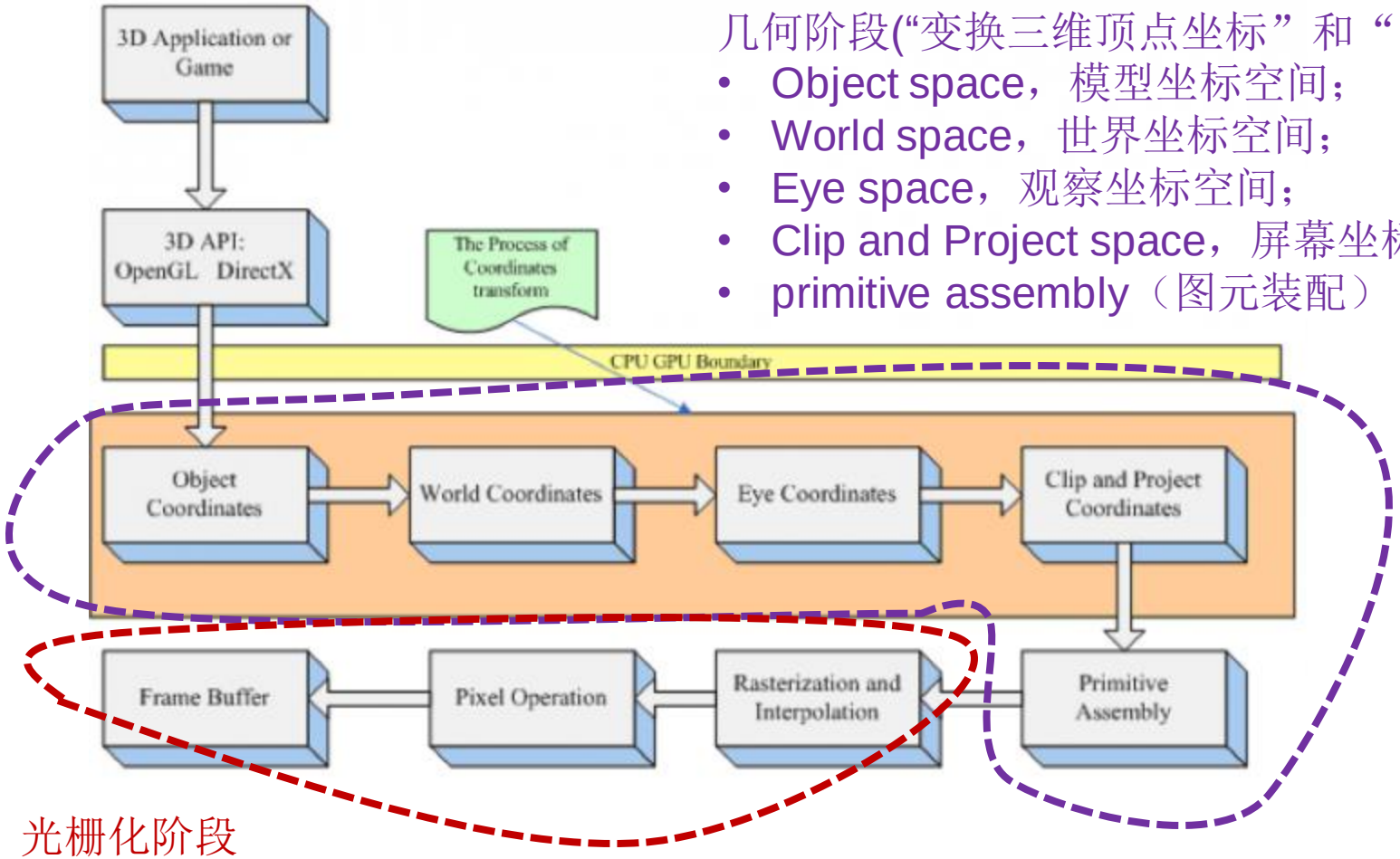
计算机如何创建图像？
渲染(rendering)

渲染(rendering)：计算机根据模型(model)创建图像的过程。
模型(model)：根据几何图元创建的物体(object)。
几何图元：包括点、直线和多边形等，它是通过顶点(vertex)指定的。



Vertex Shader (顶点着色)
Rasterization (光栅化处理)
Pixel Shader (像素着色)
Fragment shaders(片段着色)

计算机如何创建图像？
GPU在创建图像时的一般处理流程



几何阶段(“变换三维顶点坐标”和“光照计算”)

- Object space，模型坐标空间；
- World space，世界坐标空间；
- Eye space，观察坐标空间；
- Clip and Project space，屏幕坐标空间。
- primitive assembly（图元装配）

光栅化阶段

- Rasterization & Interpolation
- Pixel Operation:z-buffer(消除遮挡面);Texture operation;Blending;Filtering
- Framing

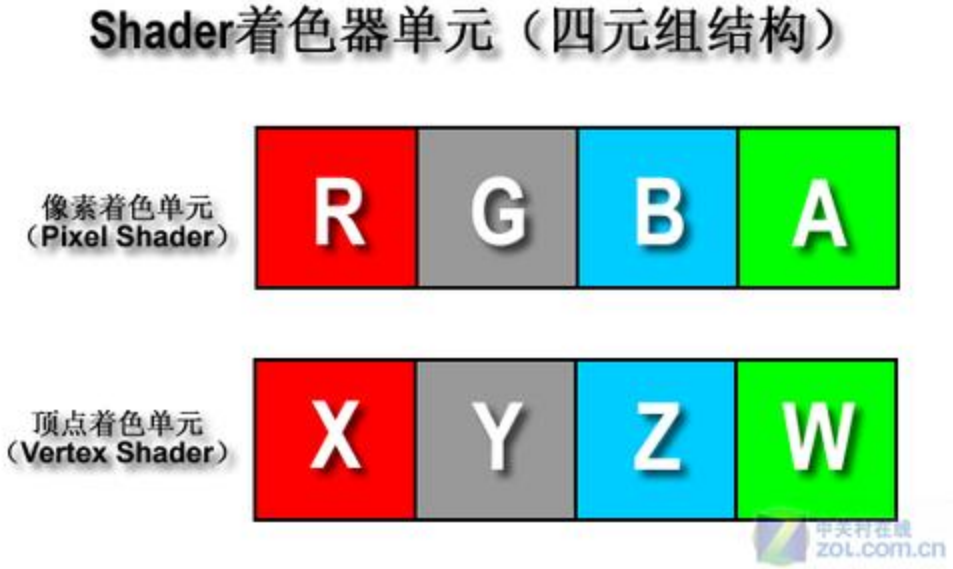
专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

创建图像过程需要什么样的硬件支持？基本电路单元：Shader(着色器单元)

在图形处理中，最常见的像素都是由RGB（红绿蓝）三种颜色构成的，加上它们共有的信息说明（Alpha），总共是4个通道。而顶点数据一般是由XYZW四个坐标构成，这样也是4个通道。在3D图形进行渲染时，其实就是改变RGBA四个通道或者XYZW四个坐标的数值。

为了一次性处理1个完整的像素渲染或几何转换，GPU的像素着色单元和顶点着色单元从一开始就被设计成为同时具备4次运算能力的算数逻辑运算器（ALU）。



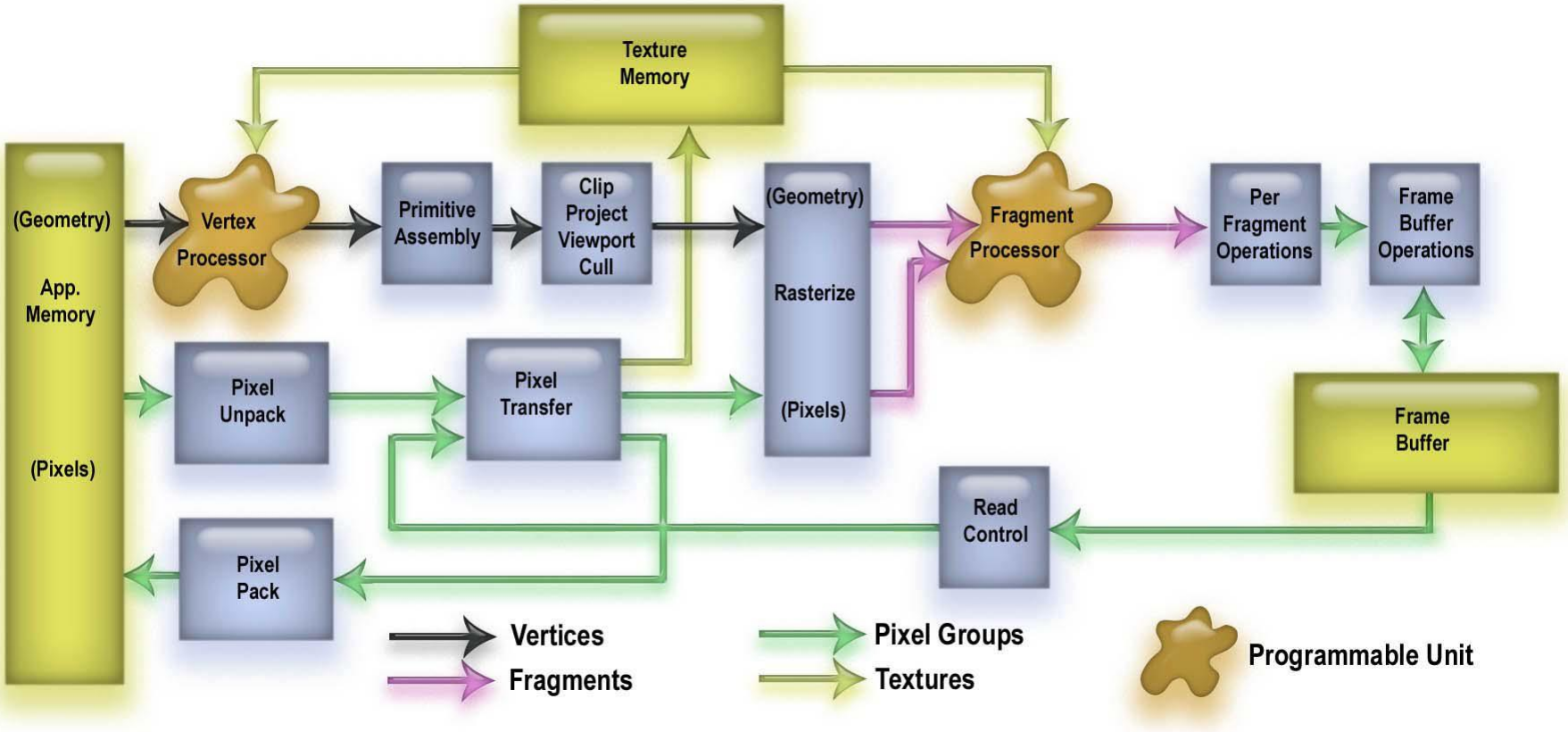
创建图像过程需要什么样的硬件支持？显卡→图形加速卡→GPU

- 图形渲染的计算要求
  - 3D图形加速运算，如z-buffering消隐，纹理映射（texture mapping），图形的坐标位置变换与光照计算（transforming & lighting）等等。这类计算的对象都是针对大量平行数据的，运算的数据量大。但面对的数据类型单一，单精度浮点占到其处理数据的绝大多数。
- 基本的显卡(Video card)
  - 将数字化的图像信息输出到模拟的显示设备上
- 图形加速卡
  - 能够快速的计算图形方面的计算，如绘制三角形，也具备常用图形图像格式的计算，如jpg解压、视频流解压等等，具备高级的纹理、材质、光照的计算，大大减轻主CPU的运算负担，从而“加速”了图形图像。
- GPU (Graphics Processing Unit)
  - The term GPU was popularized by Nvidia in 1999, who marketed the GeForce 256 as "the world's first GPU", a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that are capable of processing a minimum of 10 million polygons per second".

创建图像过程需要什么样的硬件支持？

GPU演进趋势：固化的电路单元→可编程的电路单元

One of the key developments in computer graphics has been the evolution of the GPU from a ***fixed*** pipeline to a ***programmable*** pipeline. The means various stages of the pipeline can be programmed, using programs called *shaders*.



History and Evolution of GPU Architecture
http://mcclanahoochie.com/blog/wp-content/uploads/2011/03/gpu-hist-paper.pdf

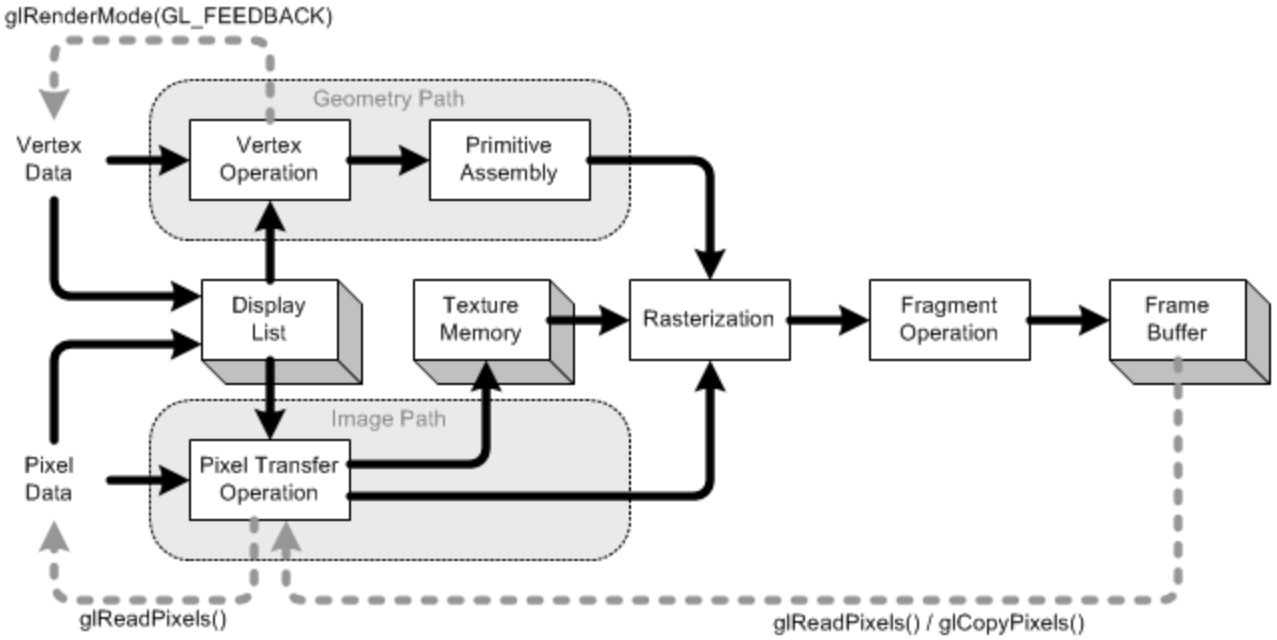
如何开发多媒体应用？

- OpenGL（Open Graphics Library）是个定义了一个跨编程语言、跨平台的编程接口规格的专业的图形程序接口。OpenGL图形库包含基本功能如下：
  - 1.建模：提供基本的点、线、多边形的绘制函数外，还提供复杂的三维物体（球、锥、多面体、茶壶等）以及复杂曲线和曲面绘制函数。
  - 2.变换：包括基本变换和投影变换。基本变换有平移、旋转、缩放、镜像四种变换，投影变换有平行投影（又称正射投影）和透视投影两种变换。
  - 3.颜色模式设置：即RGBA模式和颜色索引（Color Index）。
  - 4.光照和材质设置：OpenGL光有自发光（Emitted Light）、环境光（Ambient Light）、漫反射光（Diffuse Light）和高光（Specular Light）。材质是用光反射率来表示。场景（Scene）中物体最终反映到人眼的颜色是光的红绿蓝分量与材质红绿蓝分量的反射率相乘后形成的颜色。
  - 5:纹理映射（Texture Mapping）。利用OpenGL纹理映射功能可以十分逼真地表达物体表面细节。
  - 6:位图显示和图象增强图象功能，提供融合（Blending）、抗锯齿（反走样）（Antialiasing）和雾（fog）的特殊图象效果处理。
  - 7：双缓存动画（Double Buffering），即前台缓存和后台缓存，简言之，后台缓存计算场景、生成画面，前台缓存显示后台缓存已画好的画面。
- 1992年7月，SGI公司发布了OpenGL的1.0版本。
  - OpenGL 1.0 (1992) → …… → OpenGL 4.5 (2014)

https://www.opengl.org/

如何开发多媒体应用？
OpenGL Rendering Pipeline

OpenGL渲染管线（Rendering Pipeline）按照特定的顺序对图形信息进行处理，这些图形信息可以分为两个部分：顶点信息(坐标、法向量等)和像素信息(图像、纹理等)，图形信息最终被写入帧缓存中，存储在帧缓存中的数据(图像)，可以被应用程序获得(用于保存结果，或作为应用程序的输入等，见下图中灰色虚线)。



如何开发多媒体应用？
DirectX

- DirectX，（Direct eXtension，简称DX）是由微软公司创建的多媒体编程接口。由C++编程语言实现，遵循COM。
  - DirectX 1.0 (1995)
  - DirectX 9 (2002, shader model 2.0)
  - DirectX 9.0c (2004, shader model 3.0)
  - DirectX 10 (2006, shader model 4.0)
  - DirectX 11 (2009, GPGPU support)
  - DirectX 12 (2014.03, DirectX 12 will be essentially supported on all Fermi and later Nvidia GPUs, on AMD's GCN-based chips and on Intel's Haswell and later processors' graphics units.)

DirectX logo since DirectX 9

小结：传统的GPU

- 渲染(rendering)
  - 渲染是计算机根据模型(model)创建图像的过程。具体步骤包括：Vertex Processing、Primitive Assembly、Rasterization、Texture Mapping等。
- 显卡→图形加速卡→GPU
  - 按照2D/3D图形处理的流程设计的专用硬件电路
  - 处理流程：Pipeline(渲染管线)
  - GPU逐步朝着可编程的方向发展
- 多媒体软件开发
  - OpenGL / DirectX

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

GPU→GPGPU

GPU

- GPU（Graphic Processing Unit 图形处理器）发明的目的是应对繁杂的3D图像处理。 GPU的工作通俗来说就是完成3D图形的生成，将图形映射到相应的像素点上，对每个像素进行计算确定最终颜色并完成输出。 2000年之前，一般服务于制图、动画、游戏等电子娱乐领域（显卡/图形加速卡的功能）。

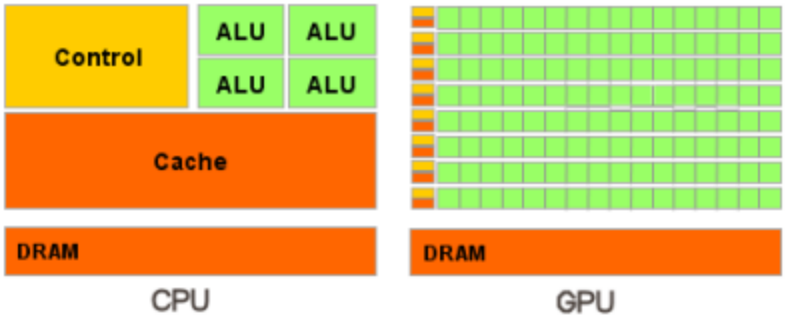
GPGPU

- 随着GPU的可编程性不断增强，可编程浮点单元已经成为GPU内部的主要运算力量，并且调用越来越方便，编程门槛不断降低。GPU的应用能力已经远远超出了图形渲染任务，利用GPU完成通用计算的研究逐渐活跃起来，将GPU用于图形渲染以外领域的计算成为GPGPU（General Purpose computing on graphics processing units，基于GPU的通用计算）。

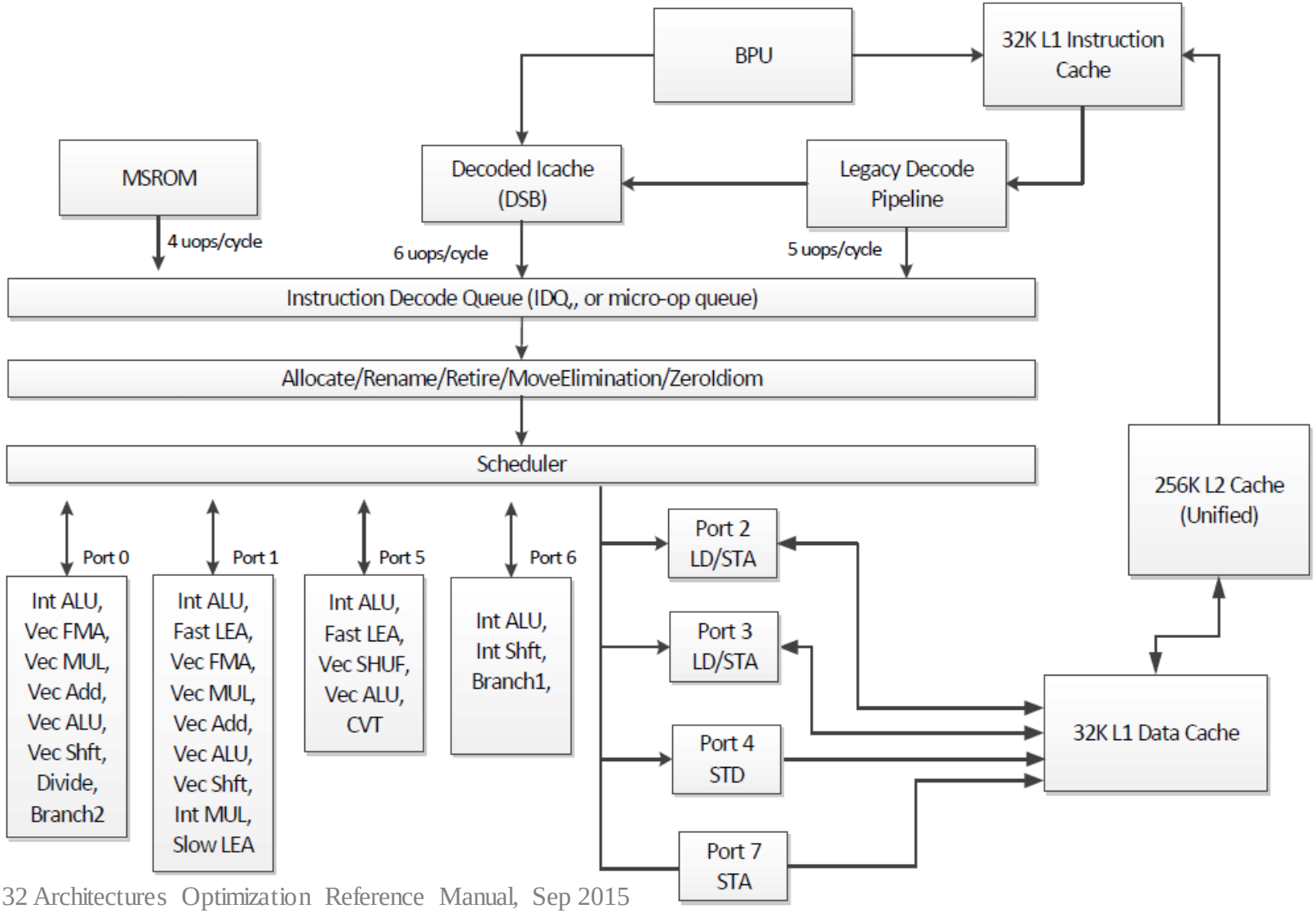
CPU vs. GPGPU

CPU和GPU设计目标不同。CPU需要很强的通用性来处理各种不同的数据类型，同时又要逻辑判断又会引入大量的分支跳转和中断的处理。这些都使得CPU的内部结构异常复杂。而GPU面对的则是类型高度统一的、相互无依赖的大规模数据和不需要被打断的纯净的计算环境。

GPU采用了数量众多的计算单元和超长的流水线，但只有非常简单的控制逻辑并省去了Cache。与CPU擅长逻辑控制和通用类型数据运算不同，GPU擅长的是大规模并发计算。



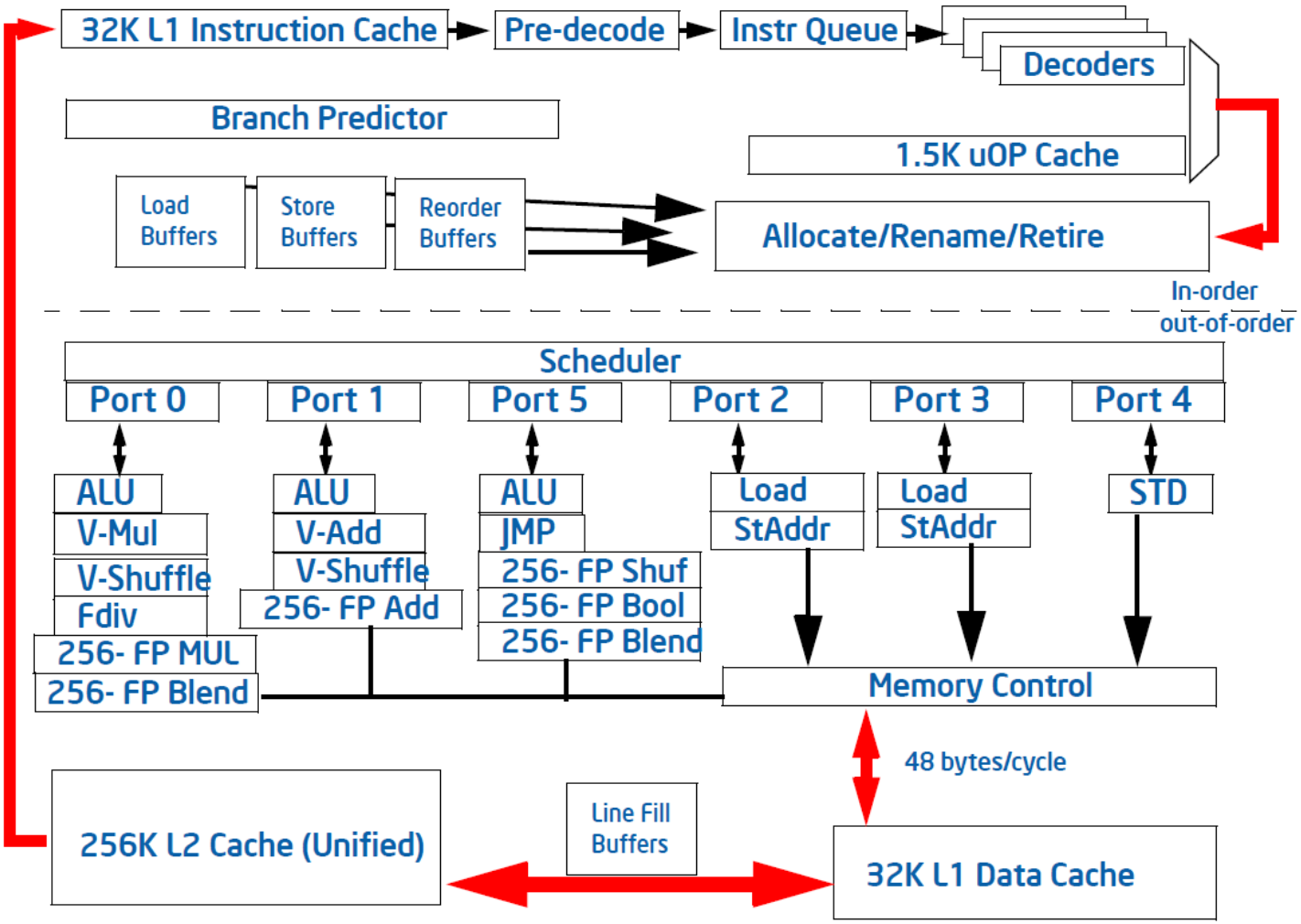
CPU示例
Intel Skylake Microarchitecture



Intel® 64 and IA-32 Architectures Optimization Reference Manual, Sep 2015
http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

CPU示例

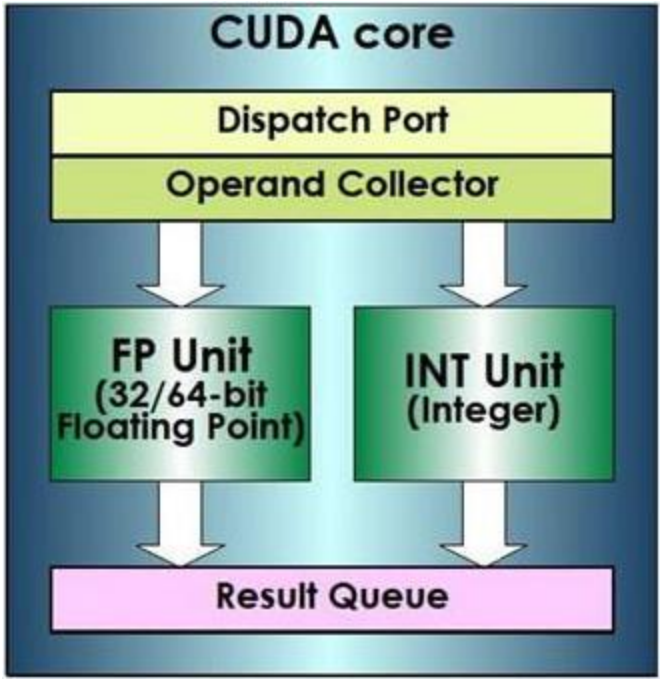
Intel microarchitecture code name Sandy Bridge Pipeline Functionality

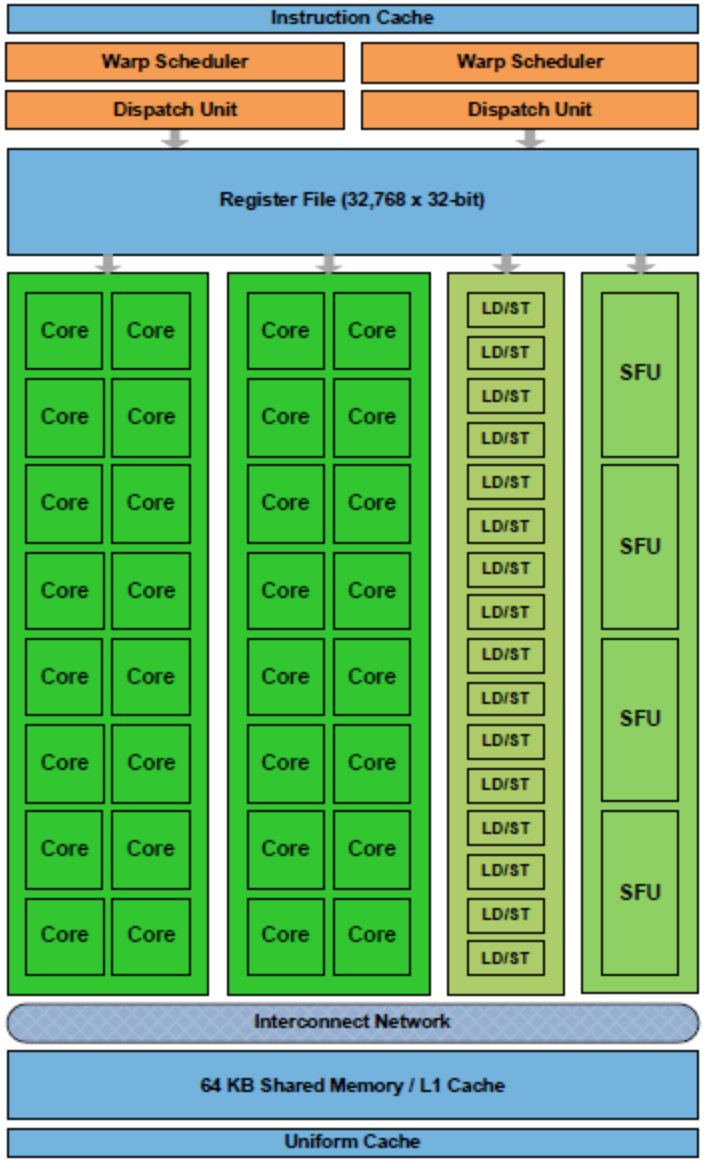


Intel® 64 and IA-32 Architectures Optimization Reference Manual, April 2012
http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

GPGPU示例
SM(Streaming Multiprocessor) Microarchitecture

2010年Nvidia推出的Fermi架构有16个SM，每个SM有32个CUDA Core，每个CUDA Core有1个整数逻辑单元（ALU）和1个浮点数单元（FPU）。





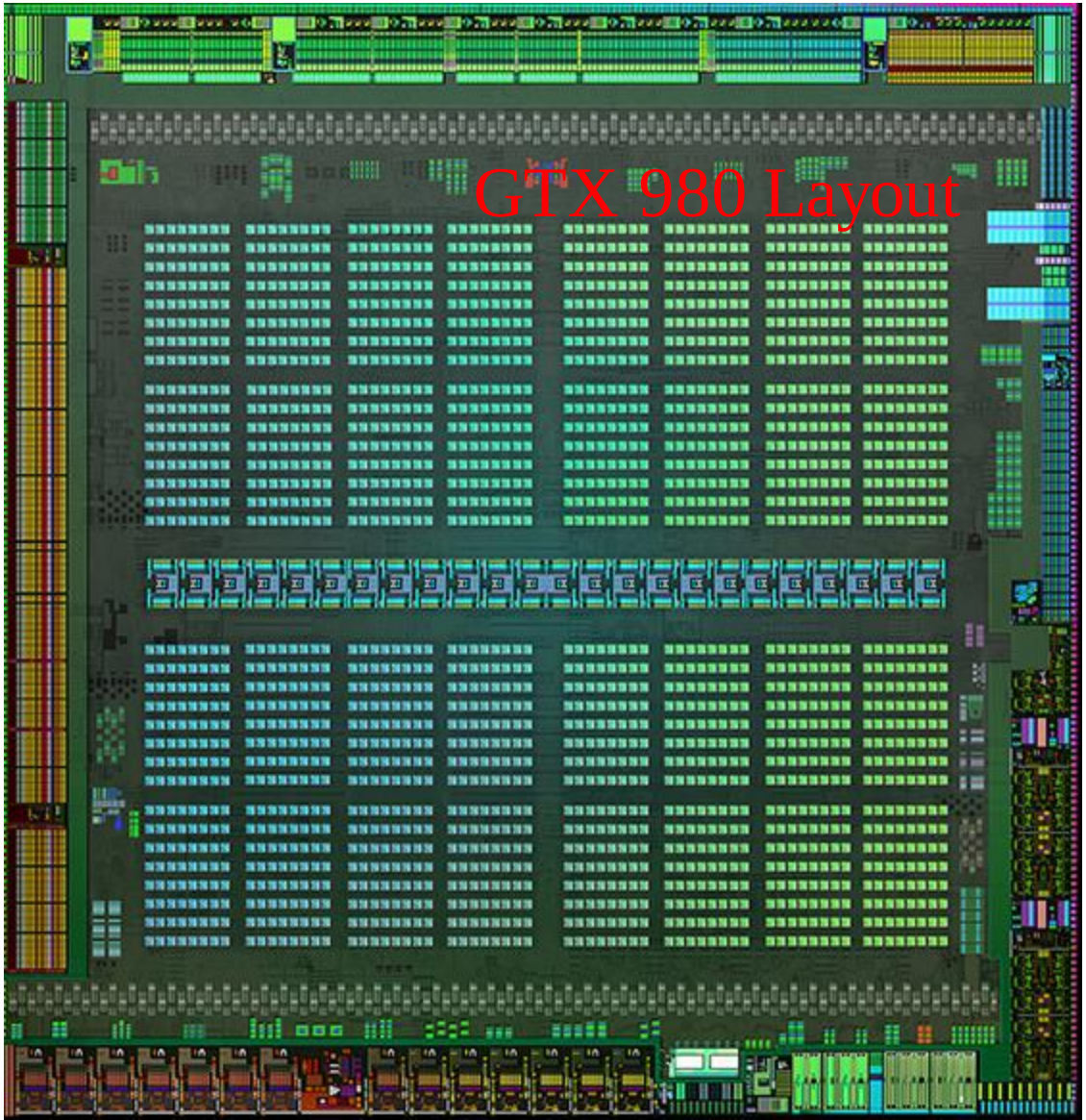
Fermi Streaming Multiprocessor (SM)

GPGPU示例
GeForce GTX 980显卡

GeForce GTX 980
核心频率 1126MHz
CUDA core 2048
理论计算能力 4.6 TFLOPs

2014年9月， NVIDIA发布GTX 980，拥有4.6T Flops的单精度浮点运算能力

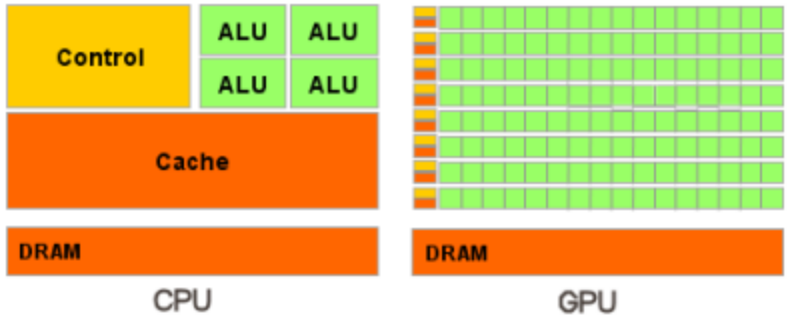
华硕GTX 980显卡，
￥4099.00(京东，2015.09)



http://www.geforce.cn/whats-new/articles/maxwell-architecture-gtx-980-970

小结：CPU vs. GPGPU

- CPU
  - 灵活多样的数据类型
  - 分支跳转等逻辑判断
  - 中断处理
  - 大容量Cache
- GPU
  - 计算单元多
  - 支持长流水线

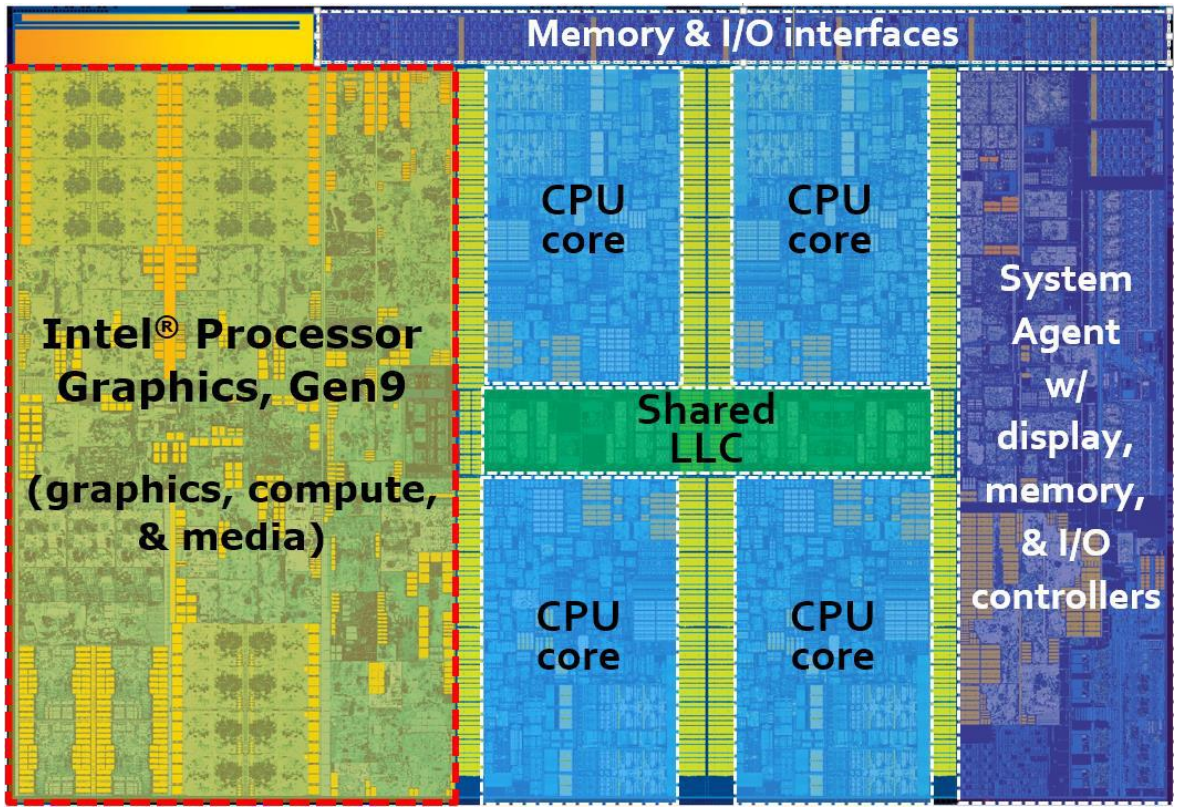


专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

I7-6700k

Figure 1: Architecture components layout for an Intel® Core™ i7 processor 6700K for desktop systems. This SoC contains 4 CPU cores, outlined in blue dashed boxes. Outlined in the red dashed box, is an Intel® HD Graphics 530. It is a one-slice instantiation of Intel processor graphics gen9 architecture.

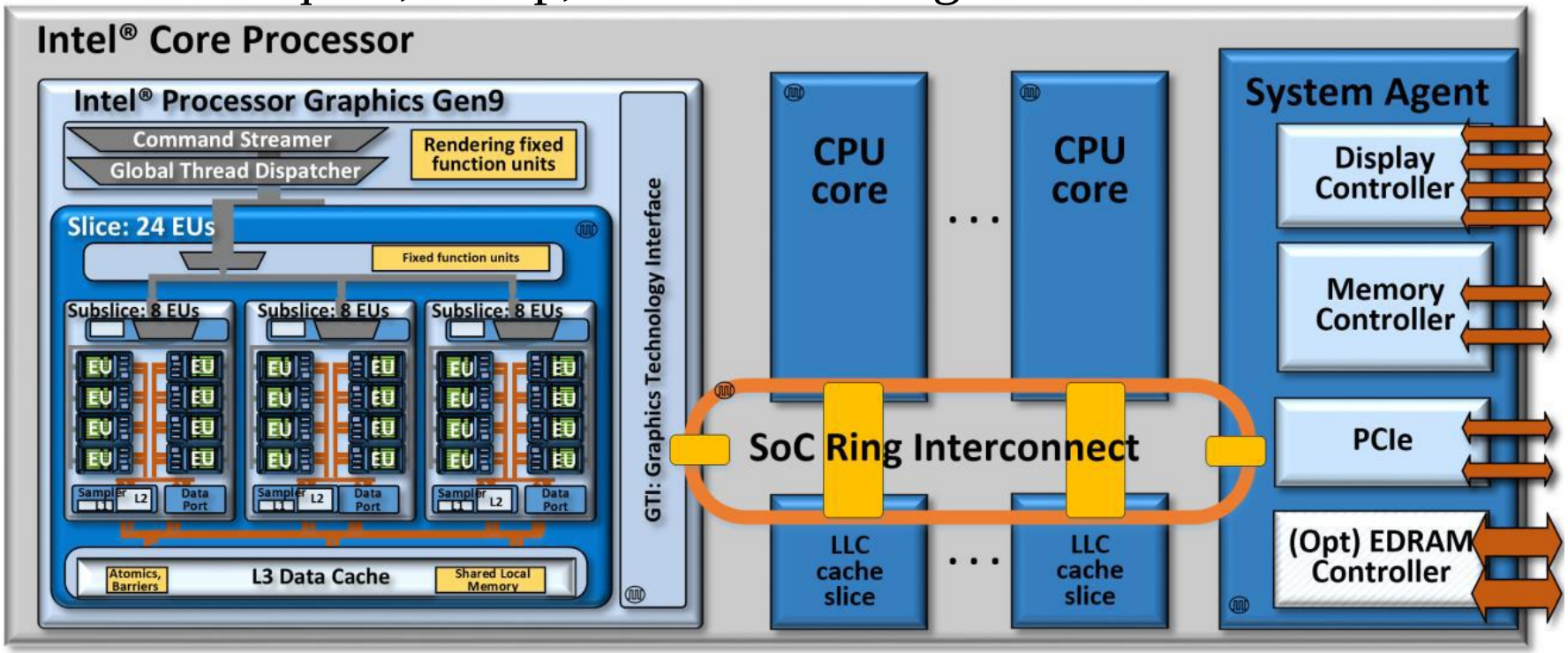


https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

An Intel® Core™ i7 processor 6700K SoC and its ring interconnect architecture

SoC及片内总线

The on-die bus between CPU cores, caches, and Intel processor graphics is a ring based topology with dedicated local interfaces for each connected "agent". This **SoC ring interconnect** is a bi-directional ring that has a 32-byte wide data bus, with separate lines for request, snoop, and acknowledge.

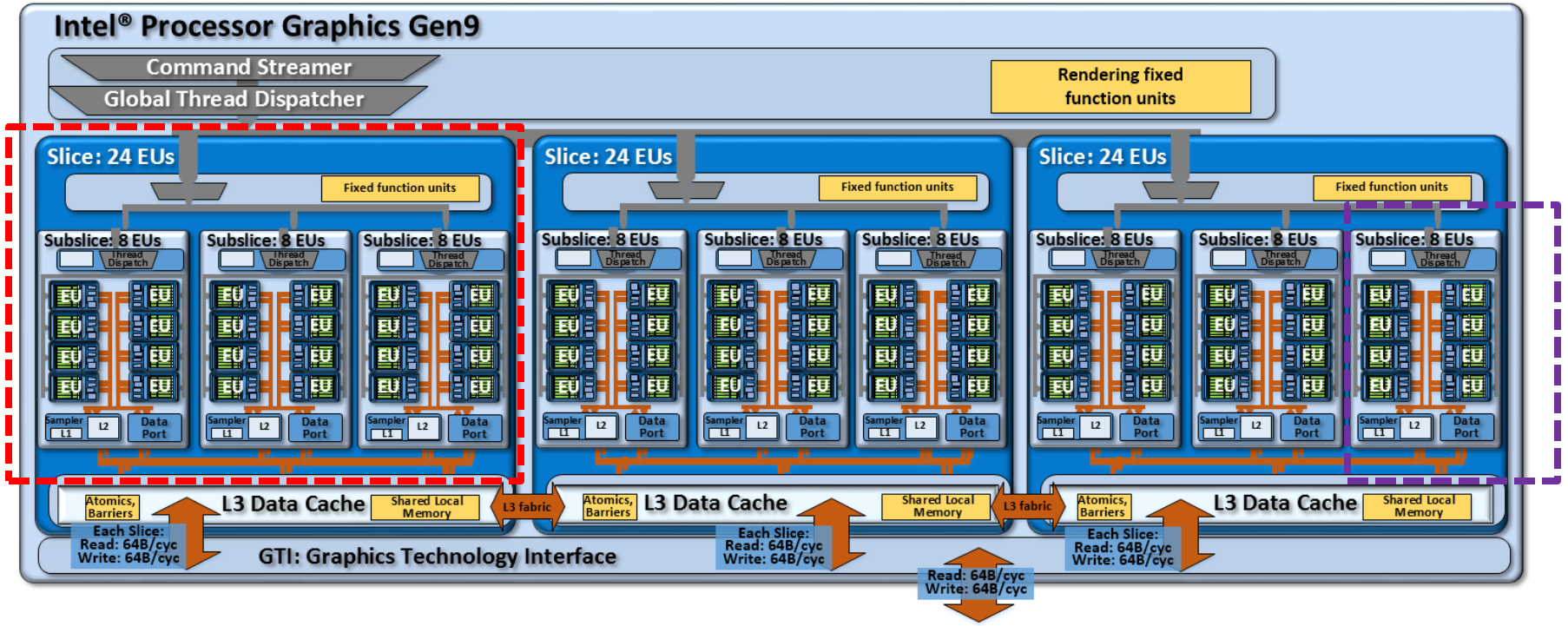


https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

- clock domain

Intel HD Graphic的构成

Intel Skylake GT4/e Graphics With 72 EUs and eDRAM



Intel HD Graphic的构成：

Execution units are clustered into groups called **subslices**. Subslices are further clustered into **slices**.

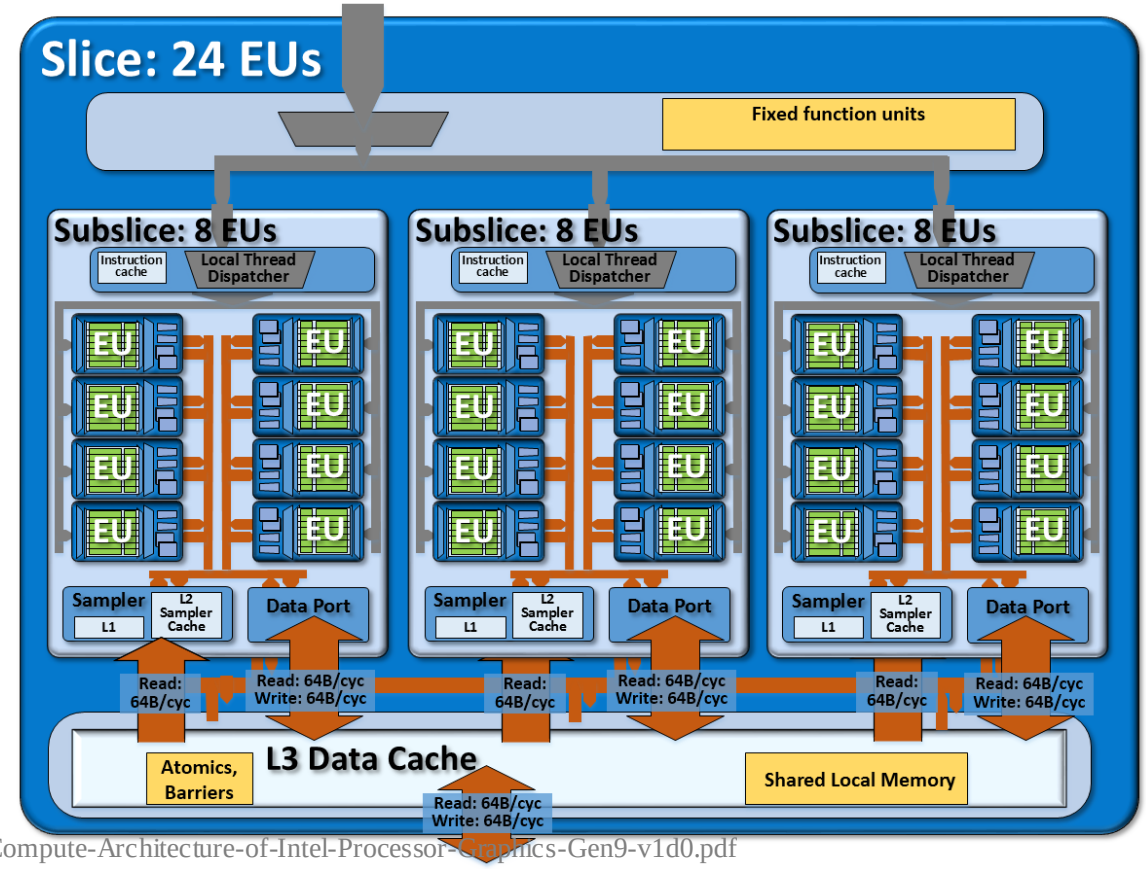
第9代显示核心最少配置一个Slice(24EUs)，最多可以配置3个Slice

https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel HD Graphic: Slice Architecture

L3 memory structure is allocated: 1) as application L3 data cache, 2) as system buffers for fixed-function pipelines, and 3) as shared local memory.
Shared local memory is a structure within the L3 complex that supports programmer-managed data for sharing among EU hardware threads within the same subslice.

This **barrier** logic is available as a hardware alternative to pure compiler-based barrier implementation approaches.
The gen9 logic can support barriers simultaneously in up to 16 active thread-groups per subslice.
Each slice also provides a rich suite of **atomic** read-modify-write memory operations.

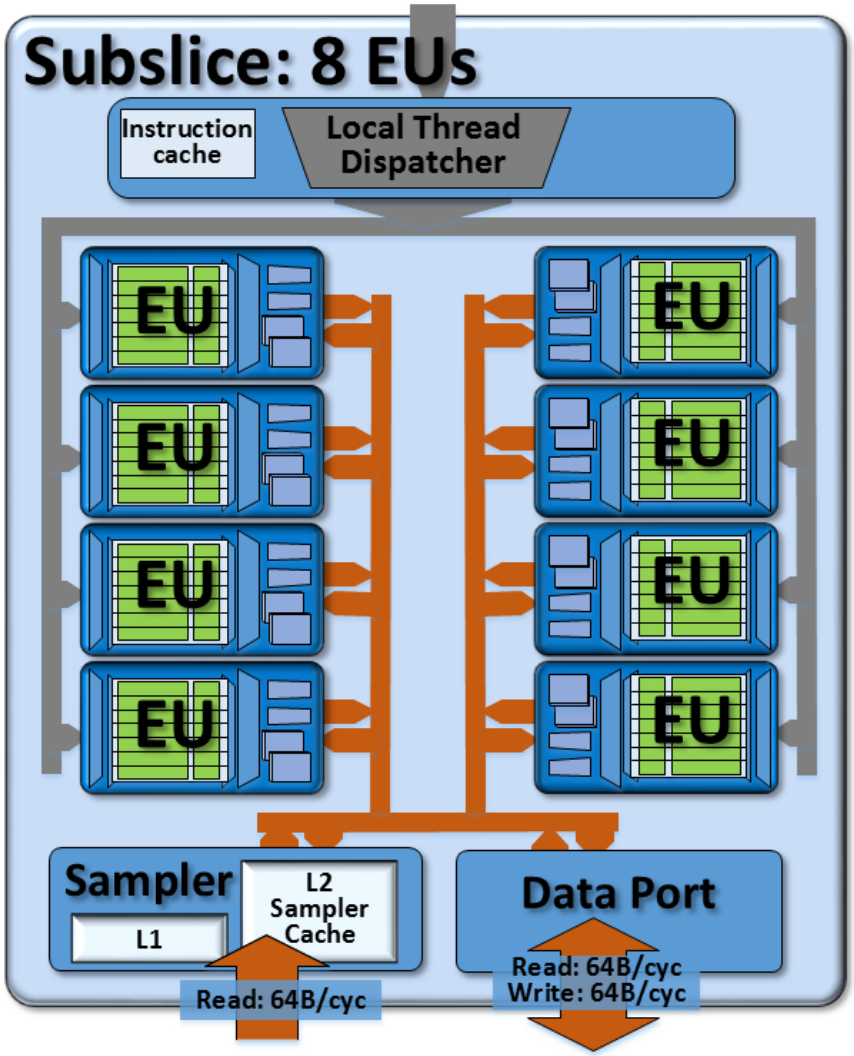


Intel HD Graphic: Subslice

Each subslice contains its **own local thread dispatcher** unit and its own supporting instruction caches. Given these 8 EUs with 7 threads each, a single subslice has dedicated hardware resources and register files for a total of **56 simultaneous threads**.

The **sampler** is a read-only memory fetch unit that may be used for sampling of tiled (or not tiled) texture and image surfaces.

The **data port** supports efficient read/write operations for a variety of general purpose buffer accesses, flexible SIMD scatter/gather operations, as well as shared local memory access.

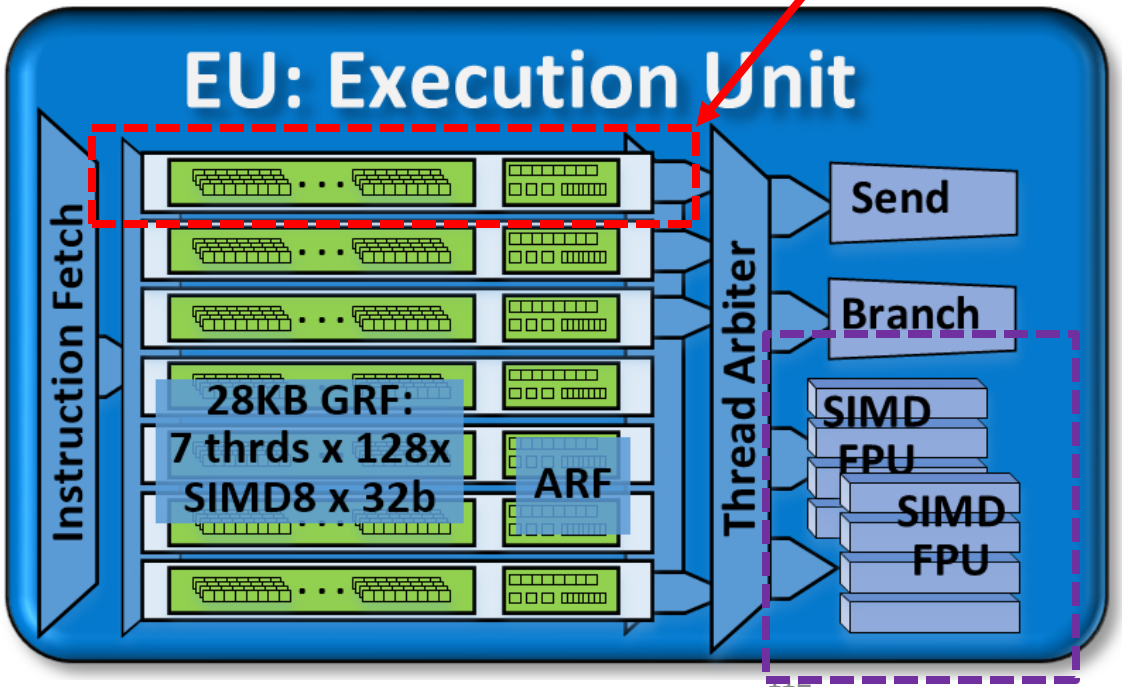


https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel HD Graphic Execution Unit (EU) Architecture

For gen9-based products, each EU thread has 128 general purpose registers. Each register stores 32 bytes, accessible as a SIMD 8-element vector of 32-bit data elements. Thus each gen9 thread has 4 Kbytes of general purpose register file (**GRF**). In the gen9 architecture, each EU has seven threads for a total of 28 Kbytes of GRF per EU.

每个thread的状态有寄存器保存



Atbiter从7个thread中选择指令并分发到4个执行部件：

- Send unit
- Branch unit
- SIMD-4 FPU
- SIMD-4 FPU

Each EU contains 2 x SIMD-4 FPUs (FP32 ALUs)

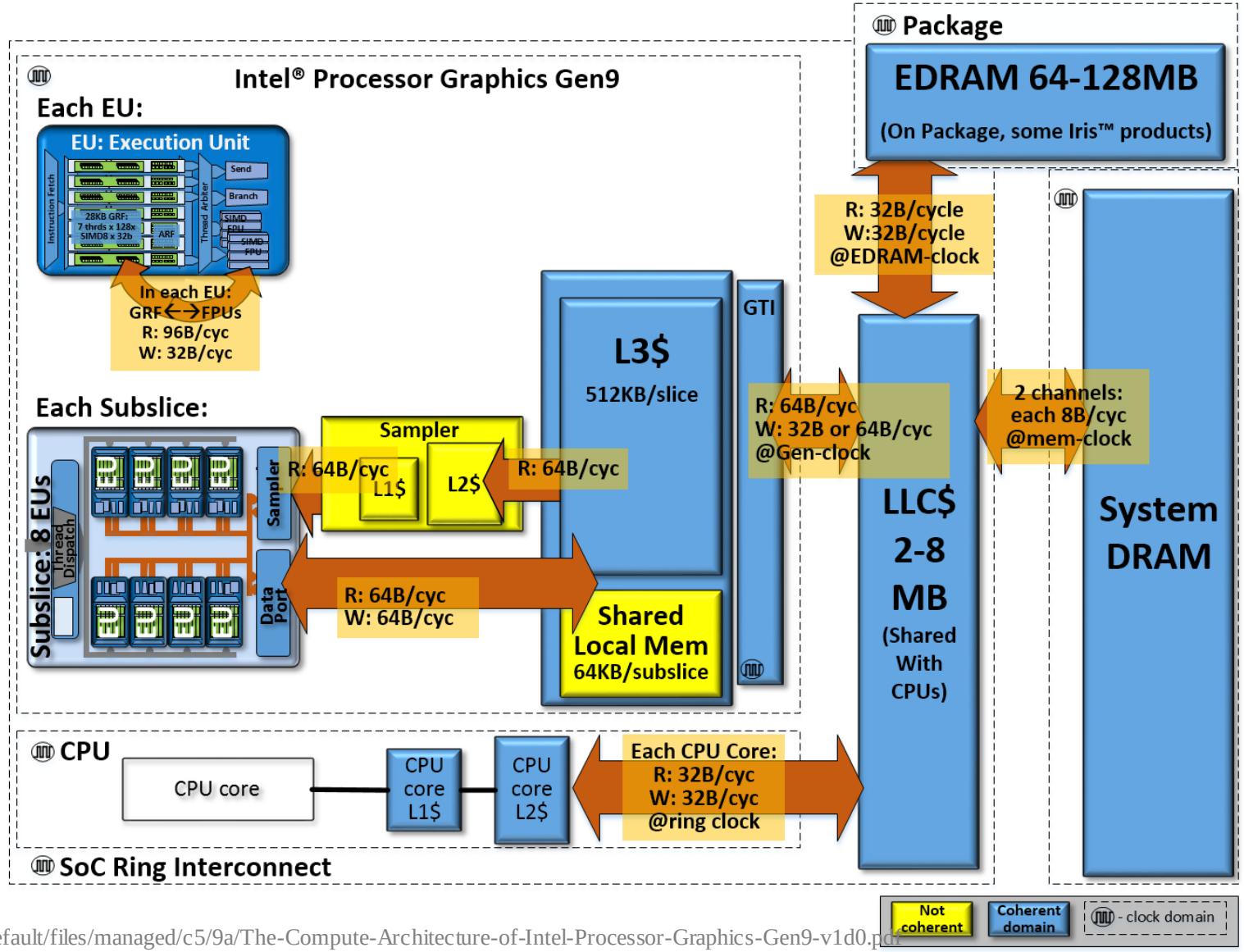
https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel HD Graphic
The EU Instruction Set Architecture (**ISA**)

- The EU Instruction Set Architecture (**ISA**) and associated general purpose register file are all designed to support a flexible SIMD width. Thus for 32-bit data types, the gen9 FPUs can be viewed as *physically* 4-wide. But the FPUs may be targeted with SIMD instructions and registers that are *logically* 1-wide, 2-wide, 4-wide, 8-wide, 16-wide, or 32-wide.
- The instruction SIMD width choice is left to the compiler or low level programmer. Differing SIMD width instructions can be issued back to back with no performance penalty.

https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel HD Graphic
memory hierarchy and theoretical peak bandwidths



https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

第6代酷睿i7-6700k(CPU:Skylake/GPU:Gen9, 2015.08) Intel® HD Graphics 530性能

At 384 FLOP/cycle and peak clock rate of 1.15 GHz, that give the Intel HD Graphics 530 a peak throughput of 441.6 single precision GFLOPS

对比
GeForce入门级GPU GT 730(690 GFLOPS, 2014.06)

AMD与CPU集成的GPU
Kaveri APUs (845 GFLOPS, 2014.01).

	Intel® HD Graphics 530	Derivation, notes
Configurations:		
Execution units (EUs)	24 EUs	8 EUs x 3 subslices x 1 slices
Hardware threads	168 threads	24 EUs x 7 threads
Concurrent kernel instances (e.g. OpenCL™ work-items or DirectX* Compute Shader "threads")	5376 instances	168 threads * SIMD-32 compile
Level-3 data cache (L3$) size	512 Kbytes	1 slice x 512 Kbytes /slice
Max shared local memory size	192 Kbytes	3 subslices x 64 Kbytes /subslice
Last level cache (LLC$) size	2-8 Mbytes	depending on product configuration
Package embedded DRAM size	n/a	
Peak Compute Throughput		
32b float FLOPS	384 FLOP/cycle	24 EUs x (2 x SIMD-4 FPU) x (MUL + ADD)
64b double float FLOPS	96 FLOP/cycle	24 EUs x SIMD-4 FPU x (MUL + ADD) x ½ throughput
32b integer IOPS	192 IOP/cycle	24 EUs x (2 x SIMD-4 FPU) x (ADD)

规划中有72 EU版本

https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel® HD Graphics趋势

6 Years of Processor Graphics

2010	2011	2012	2013	2014	2015
Iron Lake Intel® HD Graphics	**Sandy Bridge** Intel HD 3000-2000	**Ivy Bridge** Intel HD 4000-2500	**Haswell** Intel HD 5200-4200	**Broadwell** Intel HD 6200-5500	**Skylake** Intel HD 530
Intel® Core™ Processor	2nd Generation Intel Core Processor	3rd Generation Intel Core Processor	4th Generation Intel Core Processor	5th Generation Intel Core Processor	6th Generation Intel Core Processor
• 32nm		• 22nm		• 14nm	
• DirectX* 10.0	• DirectX 10.1	• DirectX 11.0	• DirectX 11.1 • DX Extensions	• DirectX 11.2	• DirectX 12.0
• Up to 10 EUs	• Up to 12 EUs	• Up to 16EUs	• Up to 40 EUs • EDRAM • Iris™ Pro, Iris™	• Up to 48 EUs • EDRAM • Iris Pro, Iris	• Up to 72 EUs • EDRAM+ • Iris Pro, Iris
43 GFLOPS†	**130** GFLOPS†	**256** GFLOPS†	**640** GFLOPS†	**768** GFLOPS†	**1152** GFLOPS†

† Peak shader FLOPS (@1GHz)

IDF15 INTEL DEVELOPER FORUM

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
    - 2D/3D图形计算的需求
    - 计算机如何创建图像？
    - 创建图像过程需要什么样的硬件支持/软件环境？
  - GPGPU (General Purpose computing on Graphics Processing Units)
    - CPU vs. GPGPU
    - Intel® HD Graphics 530
    - Intel/Nvidia/AMD运算单元的差异
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

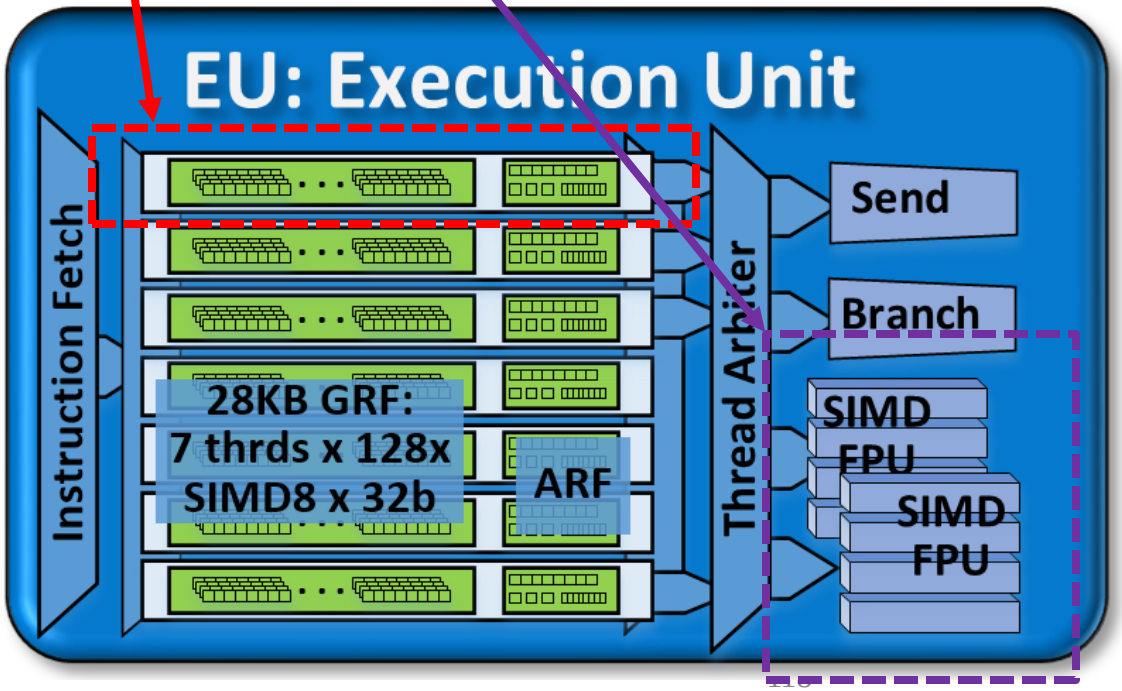
Intel GPU的最小单元
Execution Unit (EU) Architecture

GRF: general purpose register file

一个thread有128个32bytes的寄存器；共7个thread；故每个EU有28Kbytes的通用寄存器

FPU: Float Point Unit

2个SIMD-4 FPU，一个时钟周期可完成4次32bits加法和4次32bits乘法



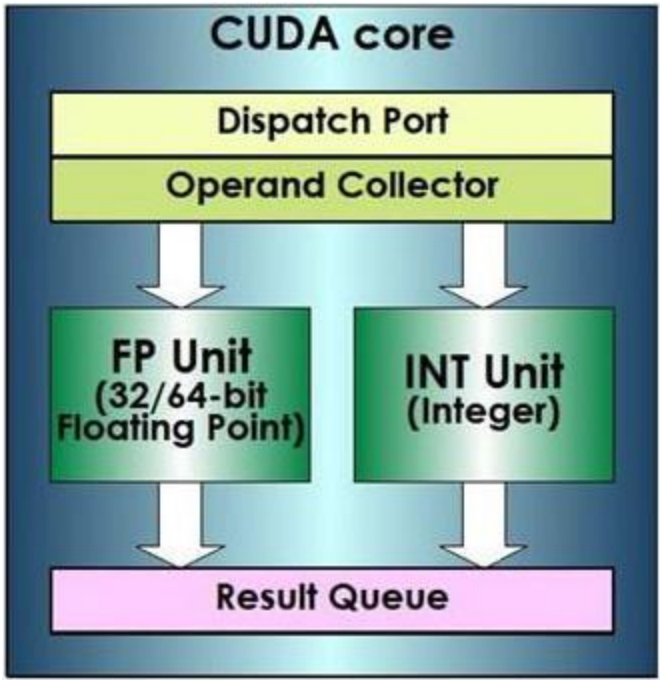
Atbiter从7个thread中选择指令并分发到4个执行部件：

- Send unit
- Branch unit
- SIMD-4 FPU
- SIMD-4 FPU

https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Nvidia GPU的最小单元：CUDA Core
Fermi架构之后统称为CUDA Core

每个CUDA Core有1个整数逻辑单元（ALU）和1个浮点数单元（FPU），在一个时钟周期内可以完成一次浮点数乘法和一次浮点数加法运算。



如GTX 980共有2048个CUDA Core，工作时钟为1126 MHz，其理论计算能力为 $1126*10^6*2048=$ **4.6 TFLOPs**

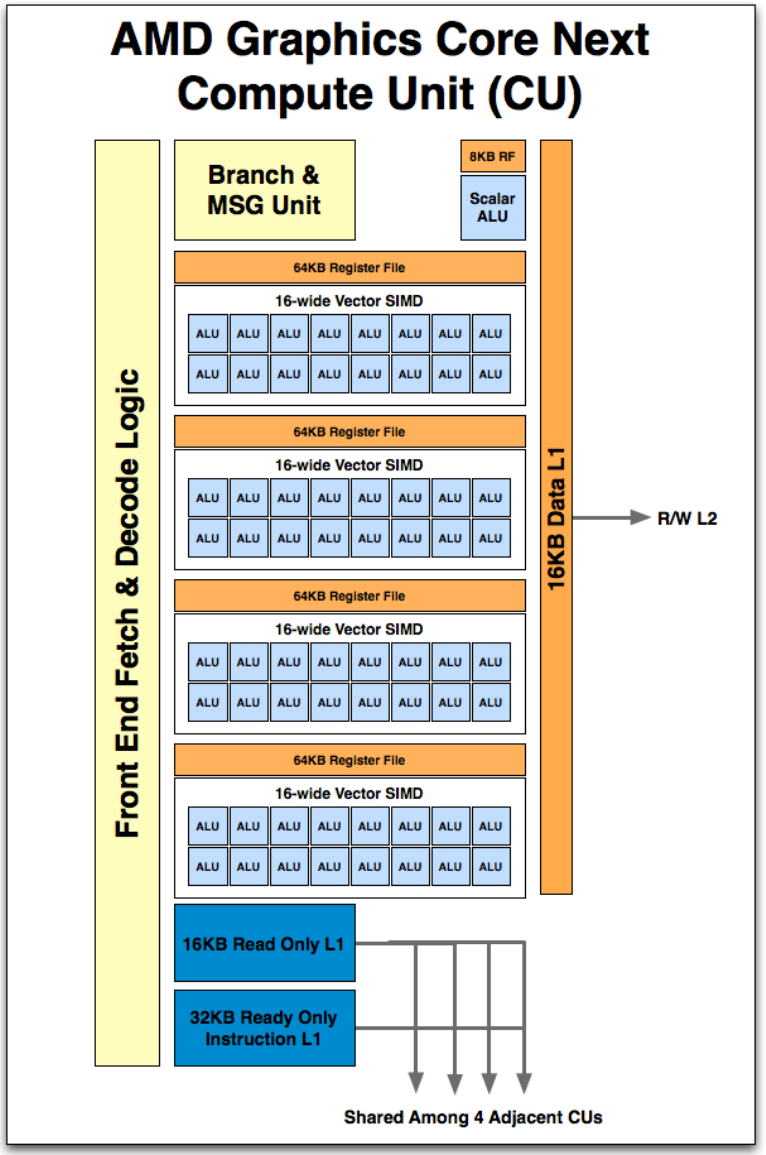
华硕GTX 980显卡，￥4099.00(京东，查询日期：2015.09)

AMD GPU最小单元：CU

AMD的GCN(Graphics Core Next)架构中的Compute Unit

Compute Unit

每组SIMD阵列实际上是由16个ALU矢量单元组成，这样一个CU单元就有64个ALU单元，每个单元在一个时钟周期内可以完成一次浮点数乘法和一次浮点数加法运算。

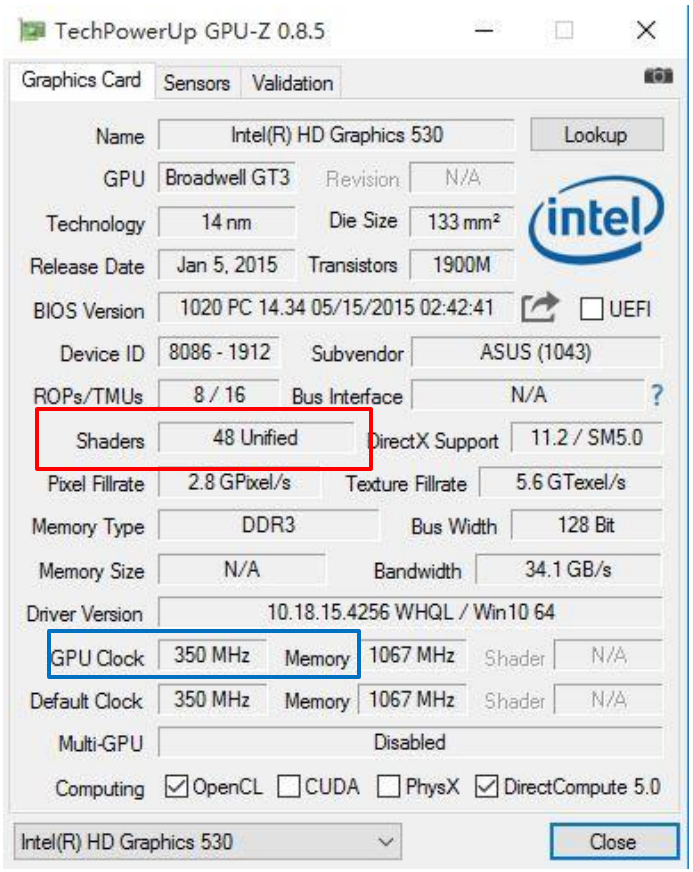


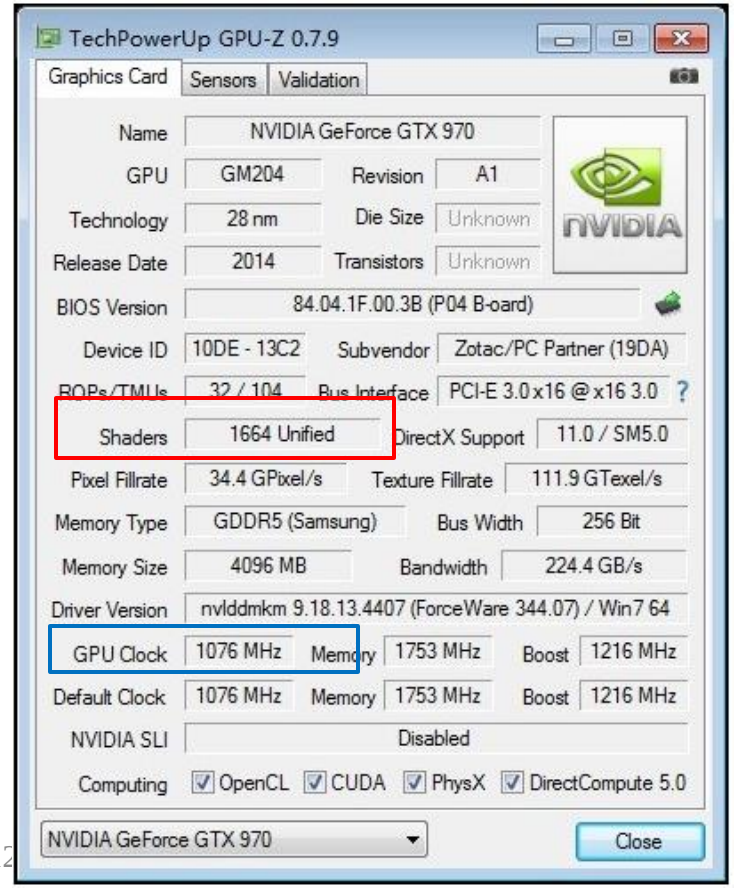
http://www.anandtech.com/show/4455/amds-graphics-core-next-preview-amd-architects-for-compute/4

GPU-Z看到的GPU信息

FLOPS = 运算单元数目 * 每Cycle的FLOPS * Clock

- Intel为EU(384FLOPS/cycle, 对于集成的24EUs的HD530: 883.2GFLOPS)
- Nvidia为CUDA Core(2FLOPS/cycle, 对于GTX970: 2.44TFLOPS)
- AMD为ALU个数??(2FLOPS/cycle)

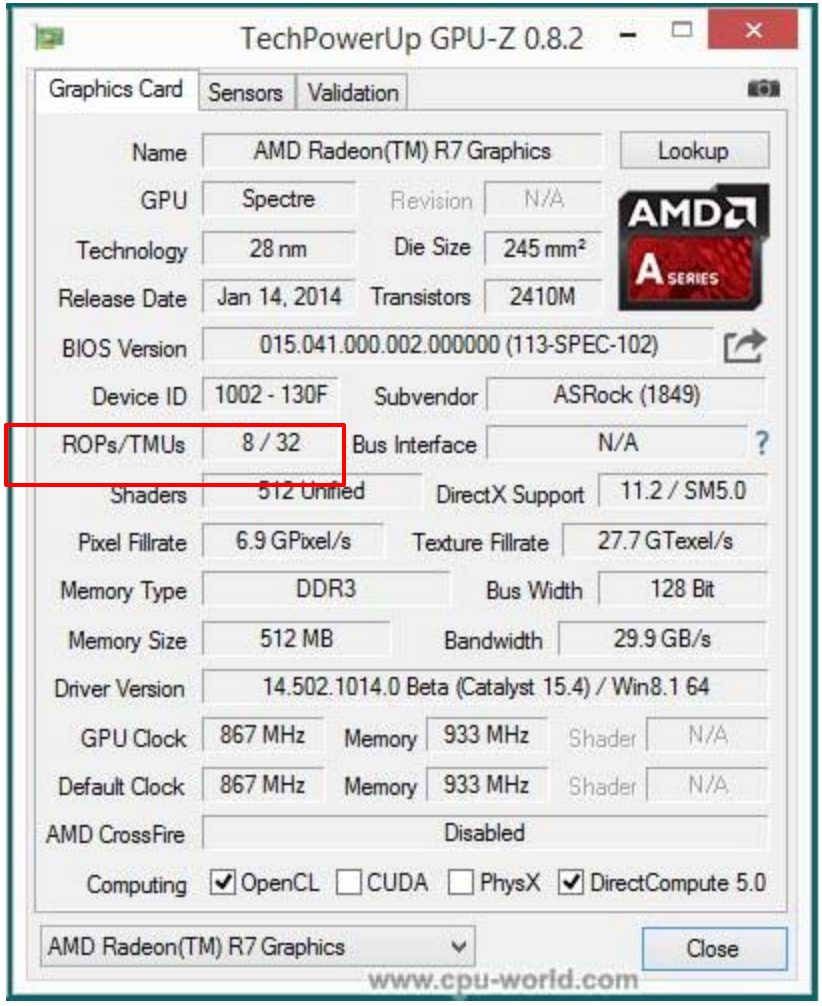




GPU-Z看到的GPU信息

固化的功能单元

TMU: Texture Mapping Unit
ROPs: Raster Operations Units



专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose Graphics Processing Units)
  - 如何使用GPGPU资源？
    - Matlab使用GPU
    - CUDA (Compute Unified Device Architecture)
    - OpenCL (Open Computing Language)
- 异构计算(Heterogeneous computing)

在MATLAB中使用GPGPU

- 2010年起，Matlab通过 Parallel Computing Toolbox 或 MATLAB Distributed Computing Server 提供对 NVIDIA 图形处理器(GPU) 的支持。
- Matlab 2015支持的GPU为CUDA-enabled NVIDIA GPU with compute capability 2.0 or higher. For releases 14a and earlier, compute capability 1.3 is sufficient.
- GPU use directly from MATLAB
  - GPU-enabled MATLAB functions such as fft, filter, and several linear algebra operations
  - GPU-enabled functions in toolboxes: Image Processing Toolbox, Communications System Toolbox, Neural Network Toolbox, Phased Array Systems Toolbox, and Signal Processing Toolbox (Learn more about GPU support for signal processing algorithms)
  - CUDA kernel integration in MATLAB applications, using only a single line of MATLAB code

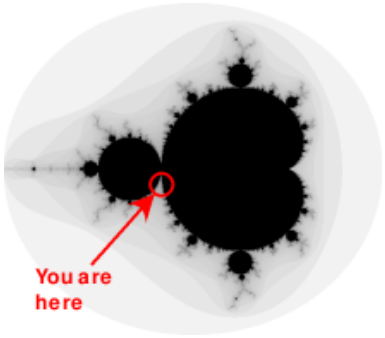
http://cn.mathworks.com/discovery/matlab-gpu.html

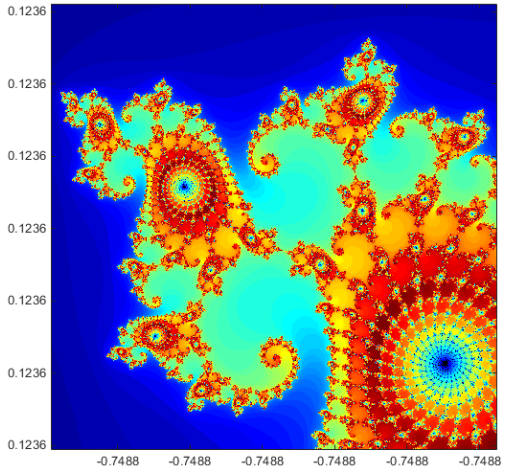
MATLAB下三种使用GPGPU的方式
Three Approaches to GPU Computing: The Mandelbrot Set

1.Convert the input data to be on the GPU using gpuArray, leaving the algorithm unchanged

2.Use arrayfun on a gpuArray input to perform the algorithm on each element of the input independently

3.Use parallel.gpu.CUDAKernel to run some existing CUDA/C++ code using MATLAB data





9.45secs (without GPU)

0.247secs(naive GPU) = 38.2x faster

0.029secs(GPU arrayfun) = 329.7x faster

0.008secs(GPU CUDAKernel) = 1119.3x faster

http://cn.mathworks.com/help/distcomp/examples/illustrating-three-approaches-to-gpu-computing-the-mandelbrot-set.html

Experiments with MATLAB (2011) http://cn.mathworks.com/moler/exm/chapters.html

MATLAB下使用GPGPU三种方式的效率

GPU中的Block & Thread

同一个 block 中的每个 thread 共享一份 share memory

每个thread都有自己的一份 register和local memory 的空间

gpuArray

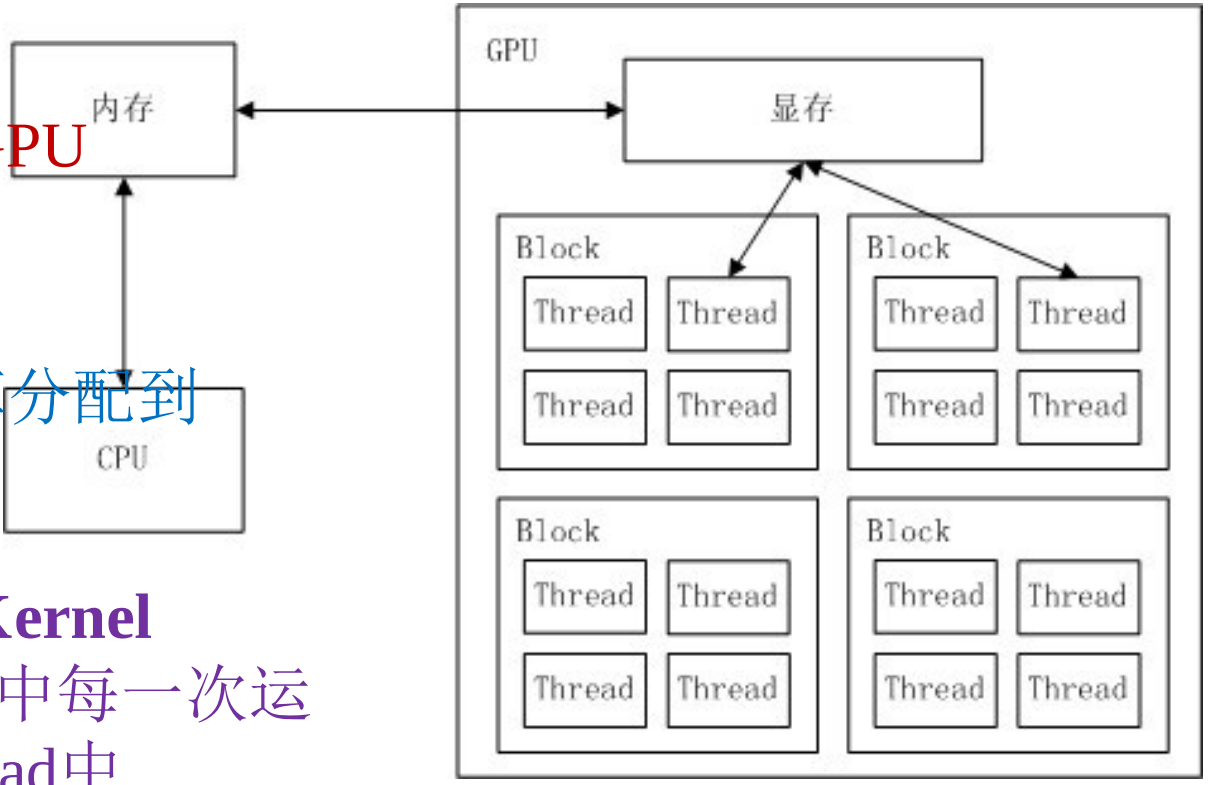
把数据拷贝到显存(GPU Memory)中

arrayfun

循环体中每一次运算分配到不同的Block中

parallel.gpu.CUDAKernel

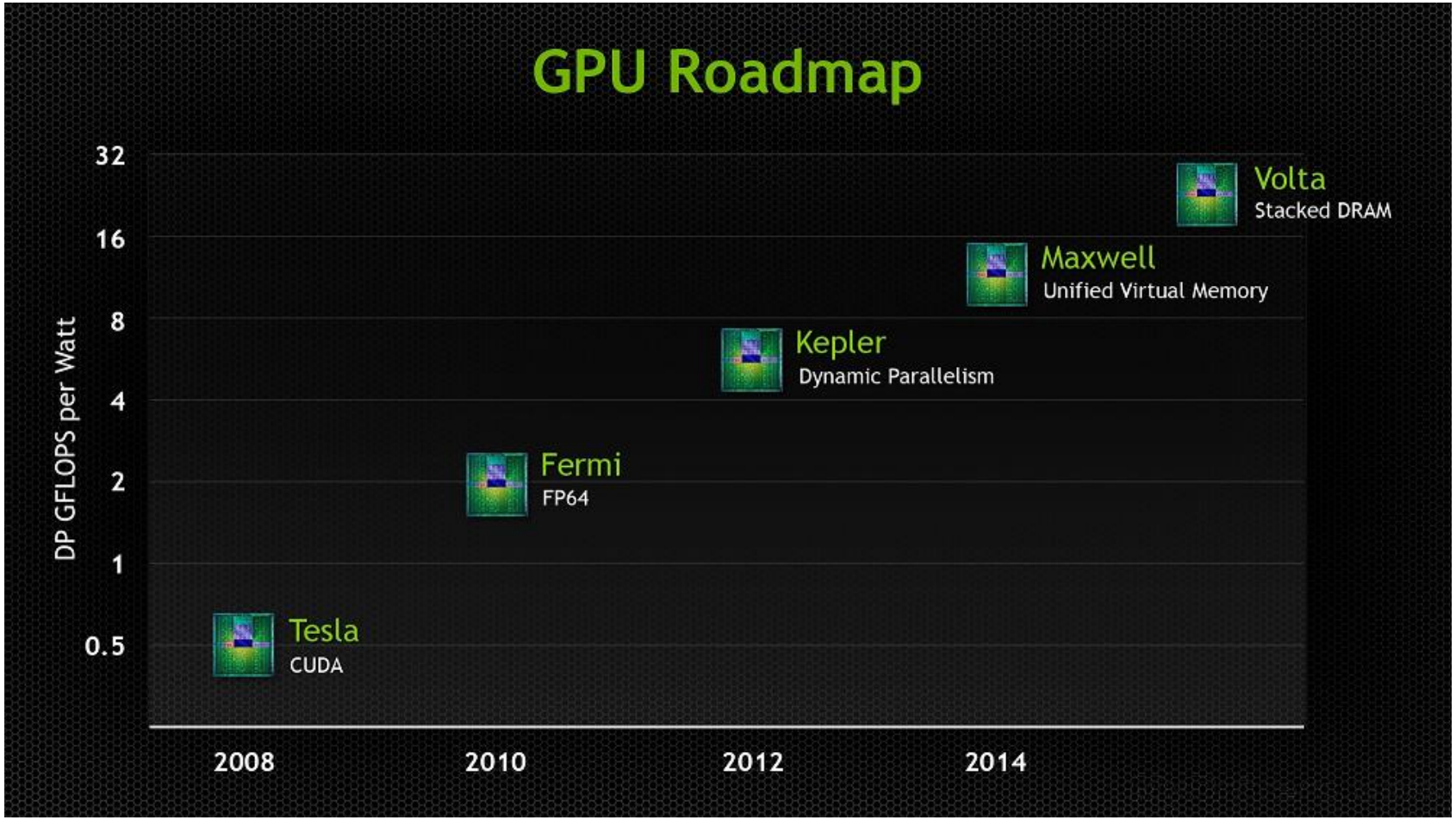
CUDA编程将循环体中每一次运算分配到不同的Thread中



通用并行计算架构CUDA (Compute Unified Device Architecture)

- CUDA(Compute Unified Device Architecture)，是一种由NVIDIA推出的通用并行计算架构，该架构包含了CUDA指令集架构（ISA）以及GPU内部的并行计算引擎。 开发人员可使用C、C++、Fortran语言来为CUDA™架构编写程序。
- 2007年6月23日，NVIDIA发布 CUDA 1.0
  - Tesla架构GPU支持 CUDA 1.0、1.1、1.2、1.3
  - Fermi架构GPU支持 CUDA 2.0、2.1
  - Kepler架构GPU支持 CUDA 3.0、3.2、3.5、3.7
  - Maxwell架构GPU支持 CUDA5.0、5.2、5.3

Nvidia的GPU架构规划



CUDA Programming Model
一个矢量加的Kernel

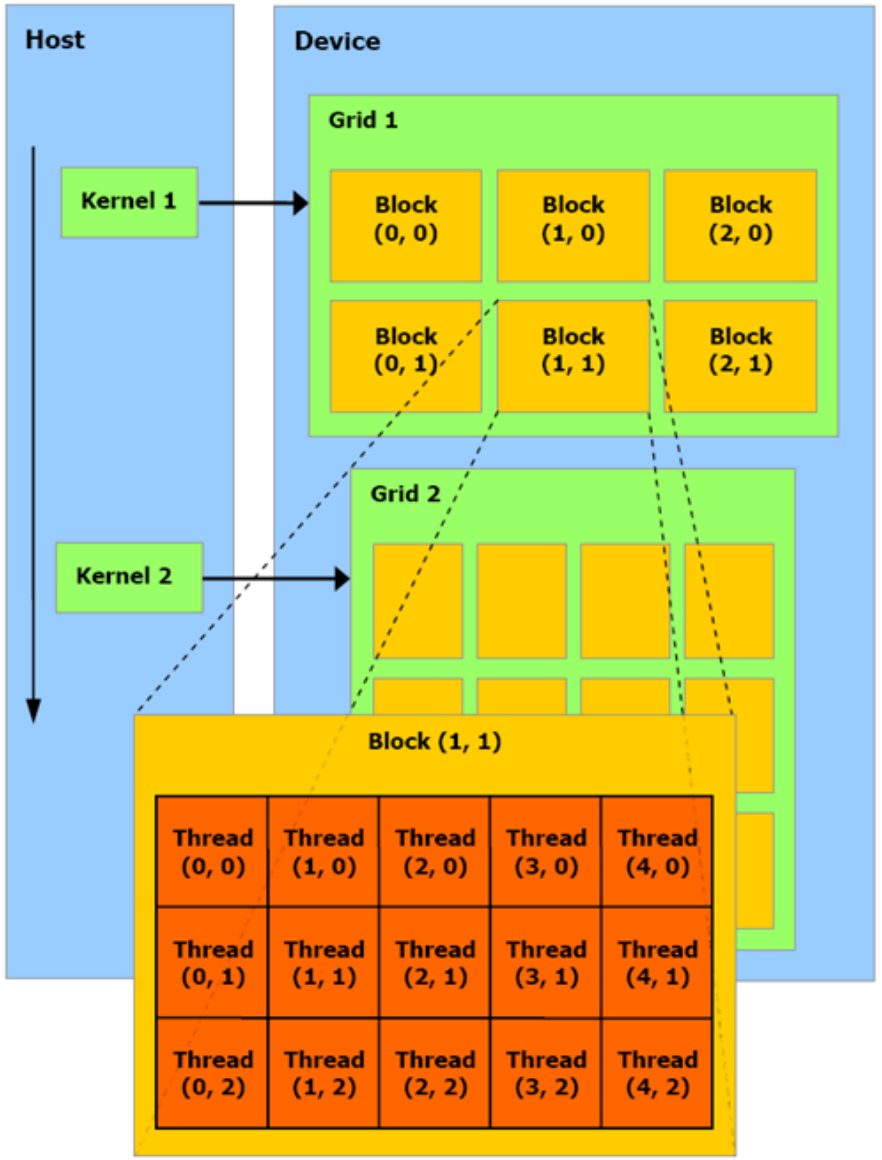
CUDA架构下，一个程序分为两个部份：host 端和 device 端。Host 端是指在 CPU 上执行的部份，而 device 端则是在GPU上执行的部份。Device 端的程序又称为 "kernel"。通常 host 端程序会将数据准备好后，复制到GPU的内存中，再由GPU执行 device 端程序，完成后再由 host 端程序将结果从GPU的内存中取回。

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C) {
 int i = threadIdx.x;
 C[i] = A[i] + B[i];
}

int main() {
 ...CPU上执行的部分…
 // Kernel invocation with N threads
 VecAdd<<<1, N>>>(A, B, C);
 ... CPU上执行的部分…
}
```

Block数目

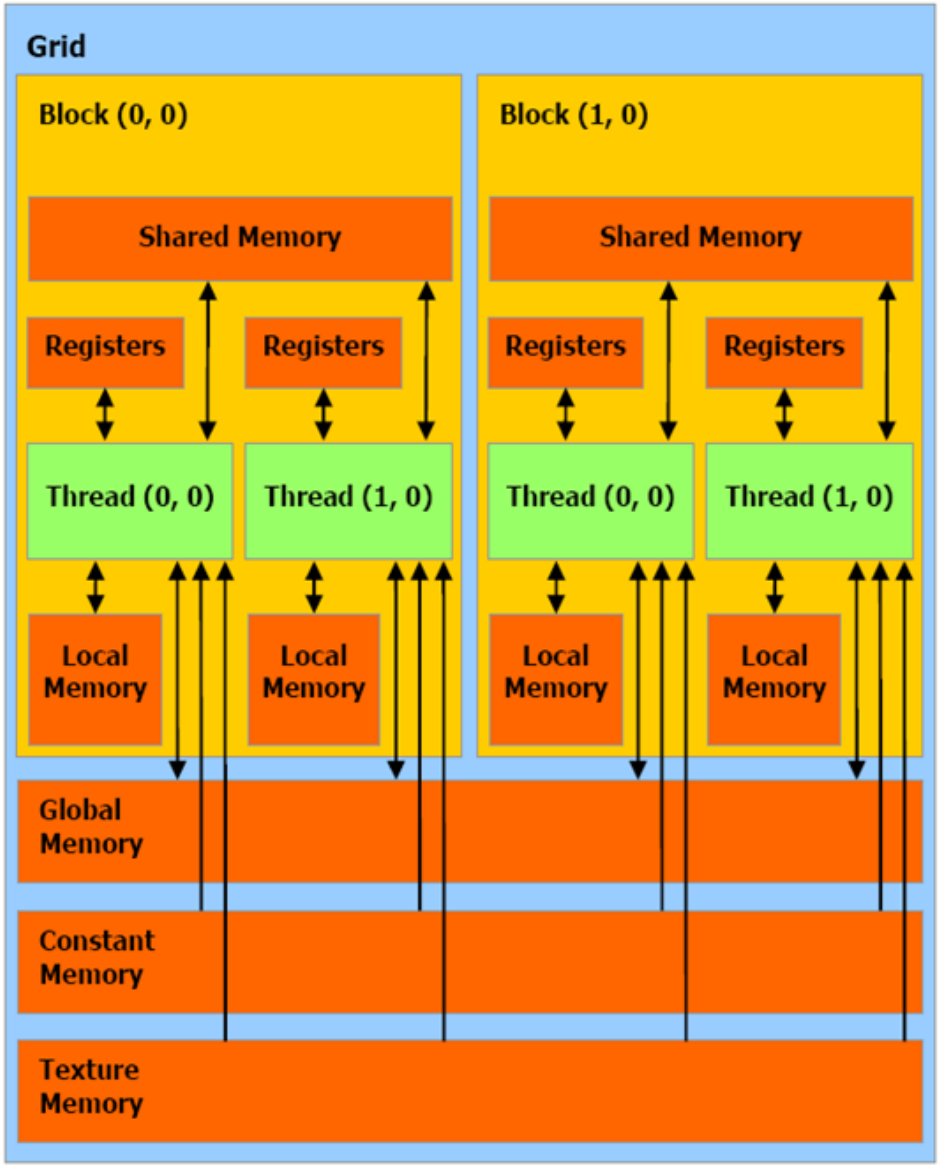
Thread数目



https://developer.nvidia.com/cuda-education
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

CUDA Programming Model
Thread→Block→Grid (Kernel)

- GPU执行时的最小单位是thread。数个thread可以组成一个block。每一个block所能包含的thread数目是有限的。不过，执行相同程序的block，可以组成grid。
- 一个block中的thread能存取同一块共享的内存，而且可以快速进行同步的动作。
- 不同block中的thread无法存取同一个共享的内存，因此无法直接互通或进行同步。
- 不同的grid则可以执行不同的程序(即kernel)。



https://developer.nvidia.com/cuda-education
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

开放运算语言 OpenCL (Open Computing Language)

◆Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors. OpenCL specifies a language (based on C99) for programming these devices and application programming interfaces (APIs) to control the platform and execute programs on the compute devices.

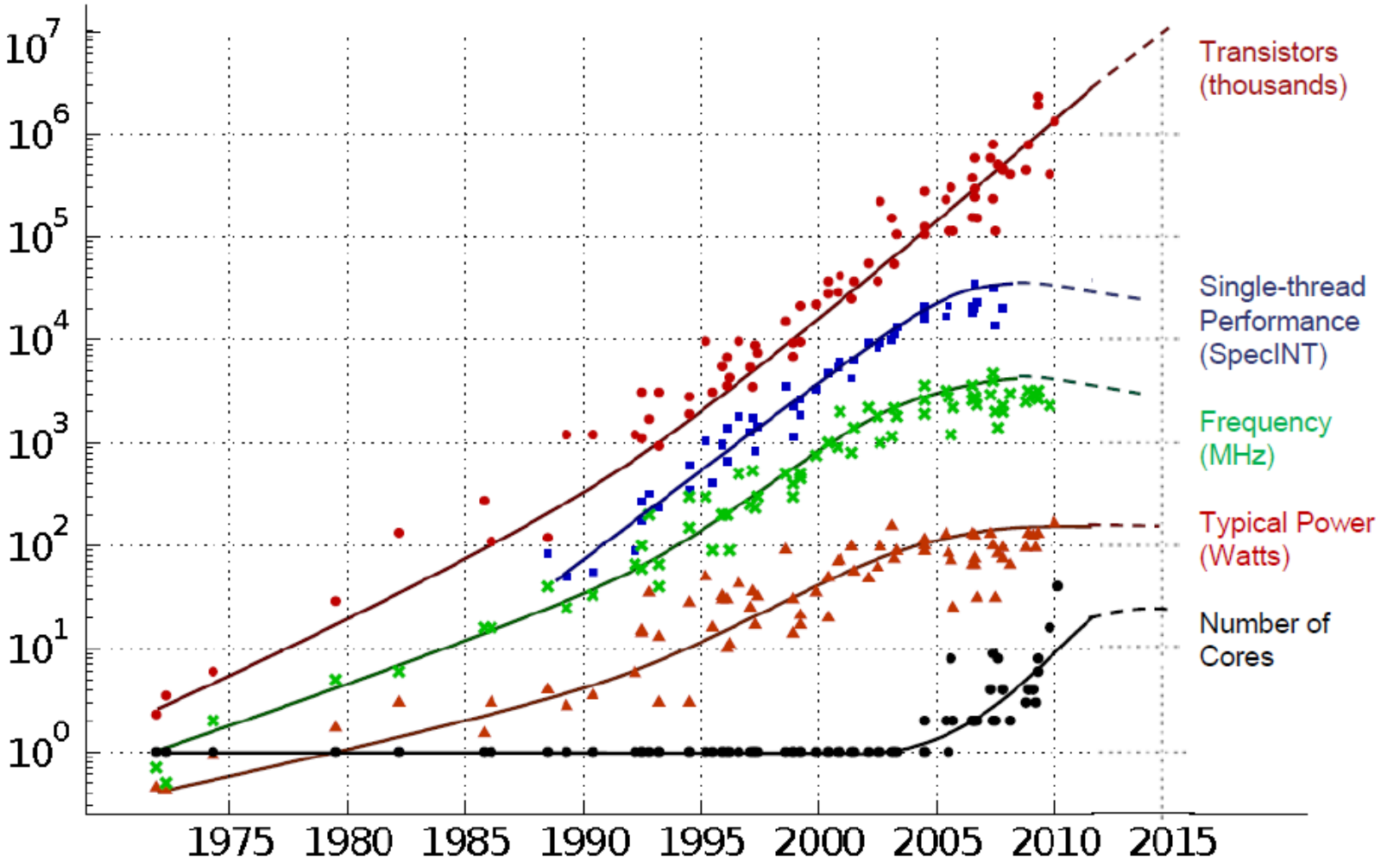
◆OpenCL最初苹果公司开发，并在与AMD，IBM，英特尔和nVIDIA技术团队的合作之下初步完善。随后，苹果将这一草案提交至Khronos Group。2008年11月18日，该工作组完成了OpenCL 1.0规范的技术细节。2010年6月14日，OpenCL 1.1 发布。2011年11月15日，OpenCL 1.2 发布。2013年11月19日，OpenCL 2.0发布。2015年1月29日， OpenCL 2.1发布。

https://www.khronos.org/opencl/
https://en.wikipedia.org/wiki/OpenCL

专题：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
- 快速地将数据送入处理器的途径
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

处理器发展趋势
集成度/性能/时钟频率/功耗/多核

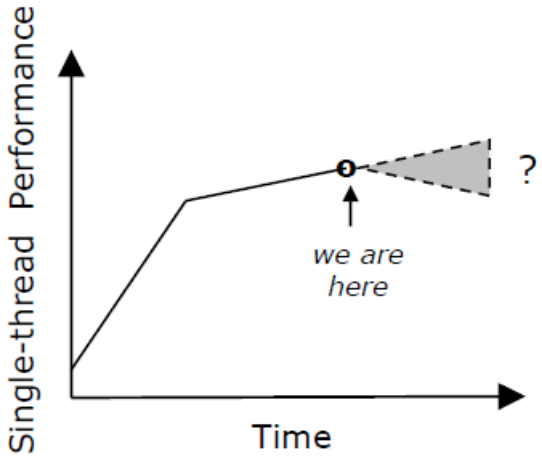


Three Eras of Processor Performance

Single-Core Era

Enabled by:
- ✓ Moore's Law
- ✓ Voltage Scaling
- ✓ MicroArchitecture

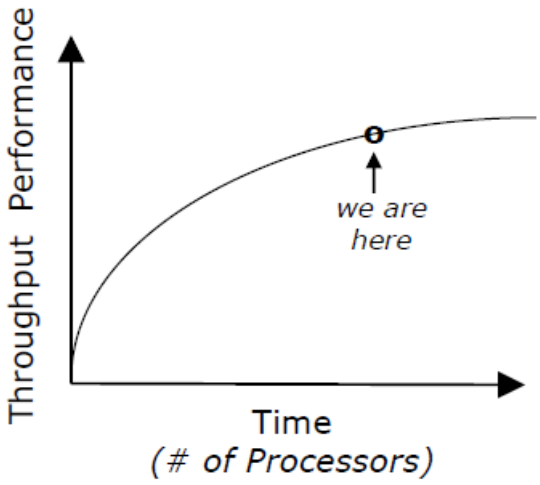
Constrained by:
- ✘ Power
- ✘ Complexity



Multi-Core Era

Enabled by:
- ✓ Moore's Law
- ✓ Desire for Throughput
- ✓ 20 years of SMP arch

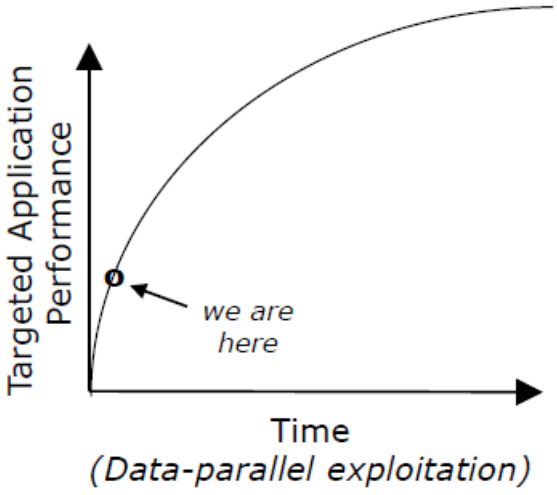
Constrained by:
- ✘ Power
- ✘ Parallel SW availability
- ✘ Scalability



Heterogeneous Systems Era

Enabled by:
- ✓ Moore's Law
- ✓ Abundant data parallelism
- ✓ Power efficient GPUs

Currently constrained by:
- ✘ Programming models
- ✘ Communication overheads



异构计算
Heterogeneous computing

◆**Heterogeneous computing** refers to systems that use more than one kind of processor. These are systems that gain performance not just by adding the same type of processors, but by adding dissimilar processors, usually incorporating specialized processing capabilities to handle particular tasks.

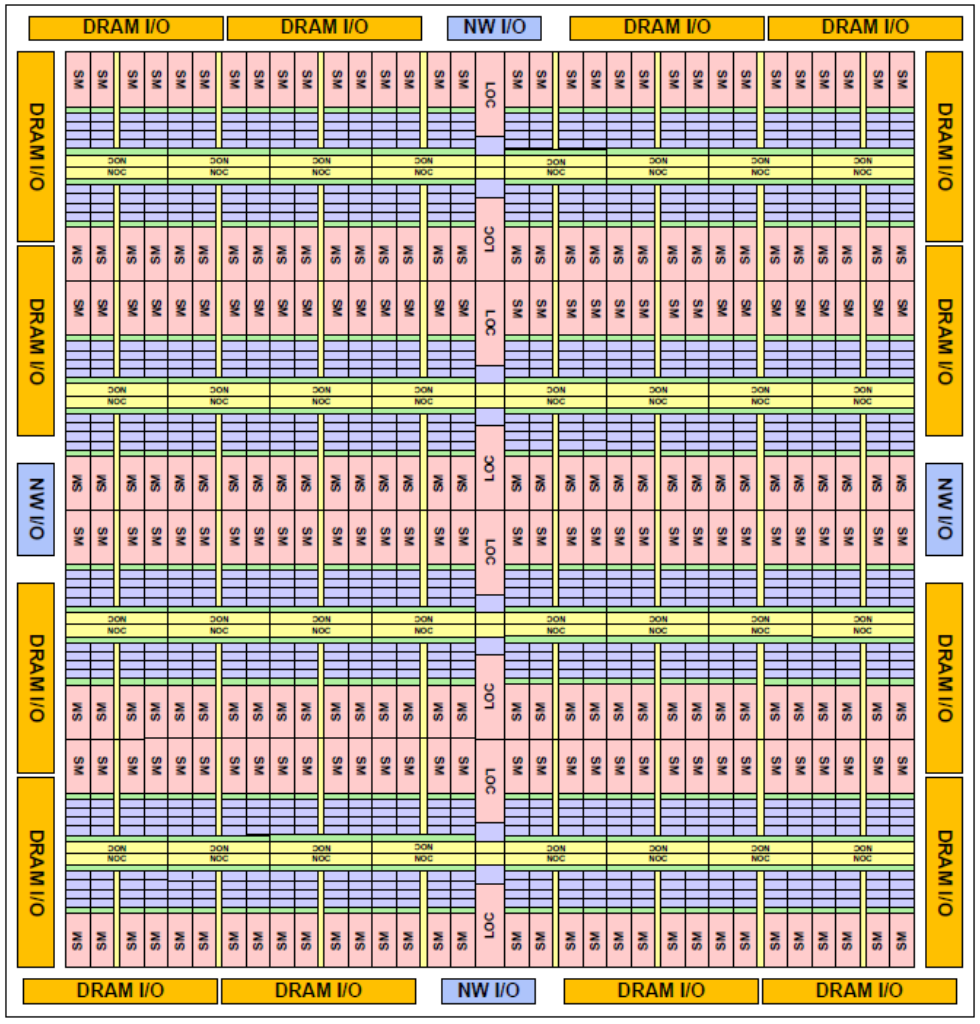
◆The level of heterogeneity in modern computing systems gradually rises as increases in chip area and further scaling of fabrication technologies allows for formerly discrete components to become integrated parts of a system-on-chip, or SoC. For example, many new processors now include built-in logic for interfacing with other devices (SATA, PCI, Ethernet, USB, RFID, Radios, UARTs, and memory controllers), as well as programmable functional units and hardware accelerators (GPUs, cryptography co-processors, programmable network processors, A/V encoders/decoders, etc.).

DARPA's Ubiquitous High-Performance Computing (UHPC)/ Exascale Projects

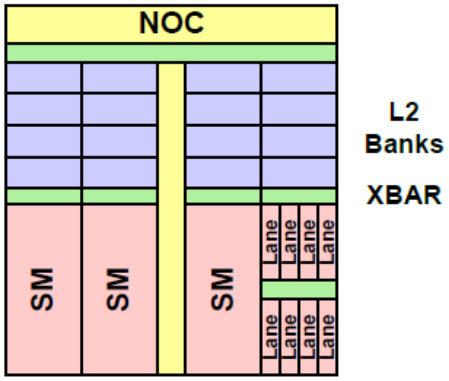
- 2010年，DARPA提出了普适高性能计算(UHPC)项目，意在开发新的计算架构和编程模型，将超级计算机系统在现有基础上提高成百上千倍，而且编程要更见简单。
- DARPA选中的UHPC项目研发带头人共有四家，分别是NVIDIA、Intel、麻省理工学院计算机科学与人工智能实验室、波士顿与桑迪亚国家实验室，预计2018年完成原型系统。
  - Intel MIC架构: Intel Many Integrated Core Architecture
    - 首款商用处理器开发代号“Knights Corner”(骑士号角)，22nm工艺制造，50多个IA架构核心，搭配Xeon作为协处理器使用
  - Echelon: NVIDIA's Extreme-Scale Computing Project
    - Echelon芯片预期含有8*256=2014个CUDA Core

Echelon芯片架构设计

NVIDIA设想每个Echelon机柜搭载32个模块，每个模块封装4个Echelon芯片



17mm



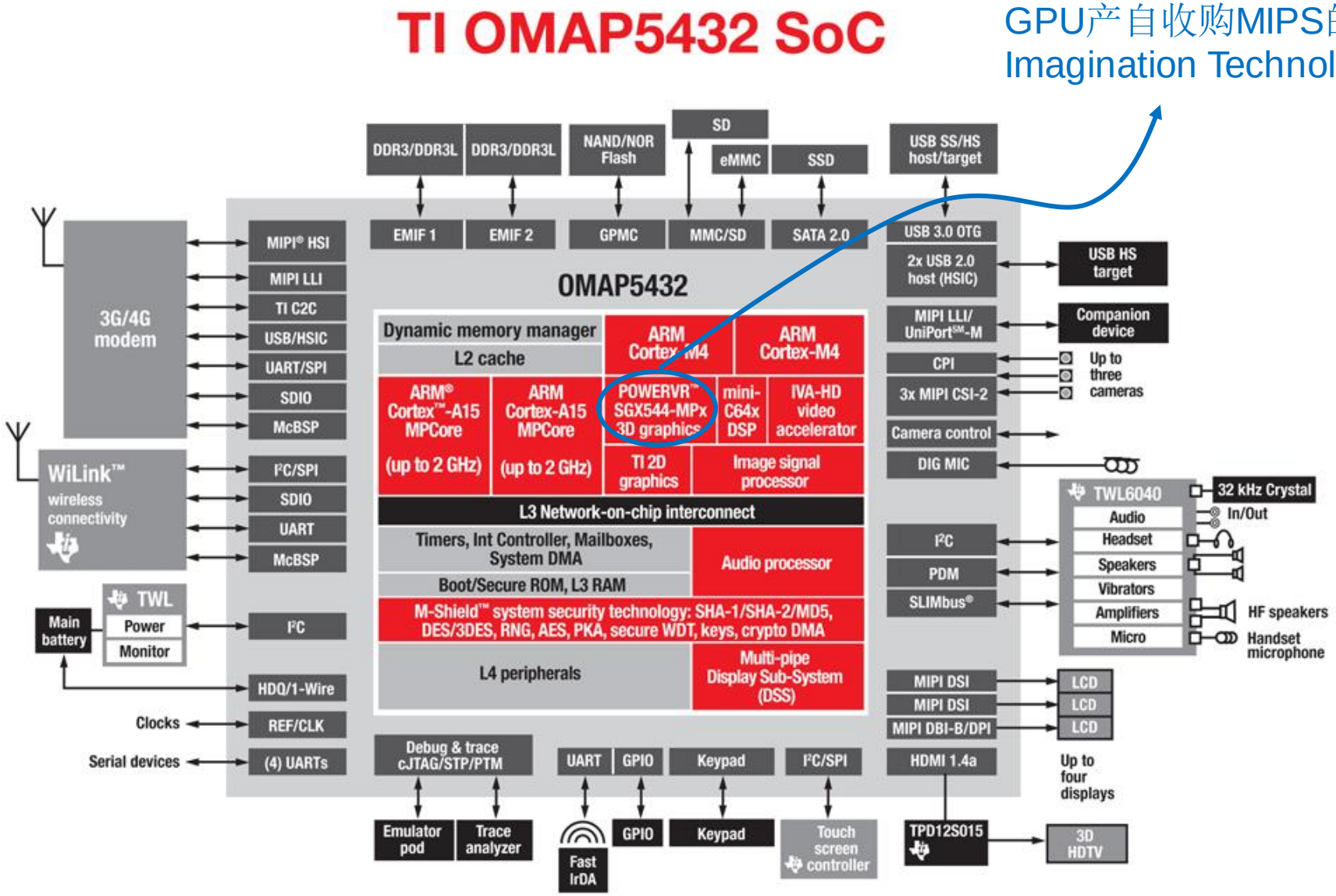
10nm process
290mm^2

256个SM模块，每个SM模块含有8个CUDA Core和独立的L0 Cache

CPU+GPU异构计算设计的超级计算机

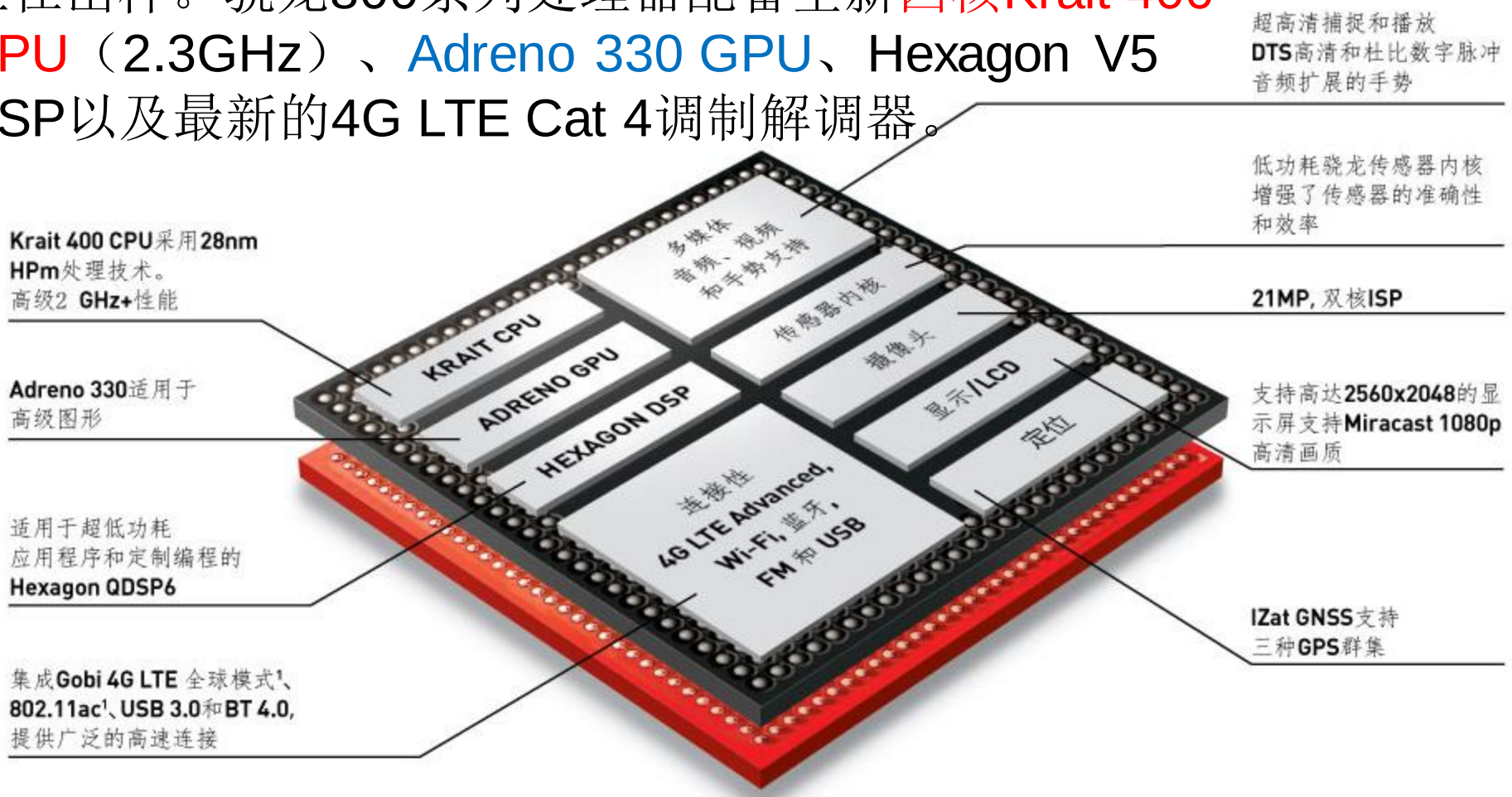
- 2009，天河一号，天津超算中心，1.206PFlops(双精度)
  - 6144个CPU（3072x2 Intel Quad Core Xeon E5540 2.53GHz/E5540 3.0GHz）；
  - 5120个GPU（2560 ATI Radeon 4870x2 575MHz）
- 2010, 天河一号A
  - 2048颗我国仿制sun公司的UltraSparc T2处理器（飞腾FT-1000）八核心处理器
  - 14336颗Intel Xeon X5670 2.93GHz六核心处理器
  - 7168块NVIDIA Tesla M2050高性能计算卡
- 天河2号，广州超算中心，54,902.4TFLOPS
  - 32000颗Ivy Bridge处理器
  - 48000个Xeon Phi(60核GPU)

ARM+DSP+GPU
德州仪器TI 的OMAP系列



ARM+DSP+GPU
高通骁龙系列处理器

2013年1月，美国高通公司宣布，骁龙800系列处理器正在出样。骁龙800系列处理器配备全新四核Krait 400 CPU（2.3GHz）、Adreno 330 GPU、Hexagon V5 DSP以及最新的4G LTE Cat 4调制解调器。



Krait架构，介于Cortex-A9和Cortex-A15之间的一个产品

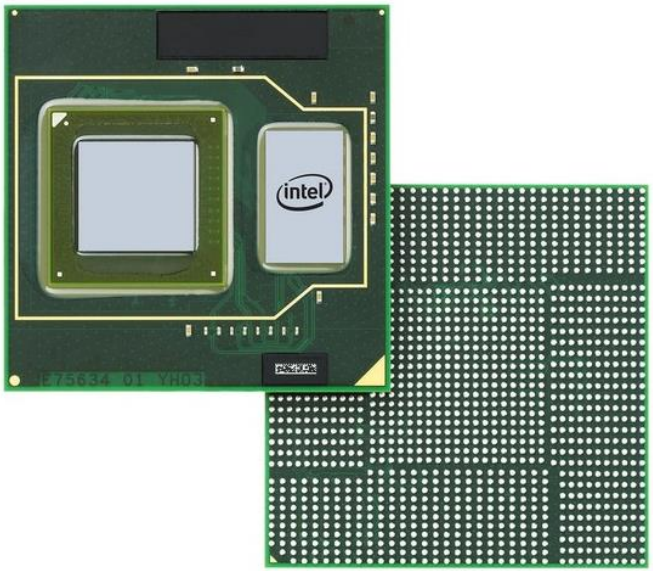
http://www.qualcomm.com/snapdragon/processors/800

CPU+FPGA的计算方式

微软Project Catapult

Catapult网络由1632台服务器构成，每台服务器由一个英特尔的Xeon处理器和一个包含Altera FPGA芯片的子卡组成，FPGA芯片被定制编程用于计算密集型工作，搜索查询的大量负荷被卸载到FPGA芯片，它处理Bing搜索算法的速度40倍于CPU。

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services
Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on



Intel CPU 整合FPGA

- 2013年2月份开始，Altera和英特尔合作让这家PC和服务器巨头为其代工
- 2014年，英特尔宣布为客户提供定制化的芯片，即将其标准处理器与针对用户服务器定制化芯片进行整合（将FPGA和服务器处理器整合在同一模块）
- 2015年6月1日，英特尔以167亿美元收购Altera

在高端FPGA芯片上集成ARM处理器

ZYNQ系列是赛灵思公司（Xilinx）推出的行业第一个可扩展处理平台，旨在为高端嵌入式应用提供所需的处理与计算性能水平。该系列四款新型器件得到了工具和IP提供商生态系统的支持，将完整的ARM® Cortex™-A9 MPCore 处理器片上系统 (SoC) 与 28nm 低功耗可编程逻辑紧密集成在一起，可以帮助系统架构师和嵌入式软件开发人员扩展、定制、优化系统，并实现系统级的差异化。

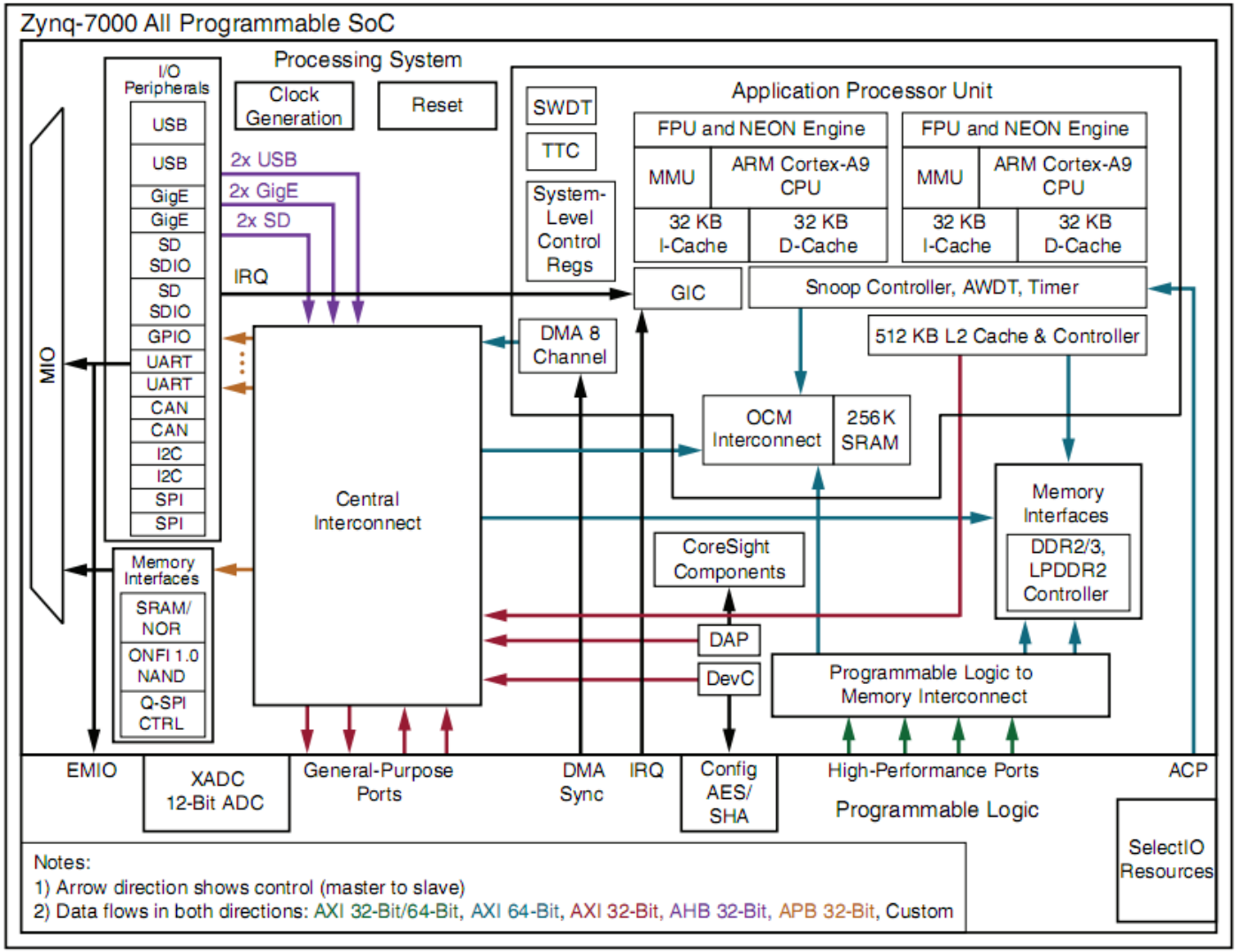


Figure 1: **Zynq-7000 All Programmable SoC Overview**

小结：异构计算

- **Heterogeneous computing** refers to systems that use more than one kind of processor.
  - GPCPU + GPGPU
  - GPCPU + GPGPU + DSP
  - GPCPU + FPGA
  - ……
- 主流软件环境
  - CUDA (Compute Unified Device Architecture)
  - OpenCL (Open Computing Language)

总结：多媒体计算→异构计算

- 多媒体计算需要什么？
- 怎么提高处理器的计算能力？
  - 协处理器集成到CPU内部
  - 通过指令集扩展提高CPU的计算能力
  - APU(CPU+GPU)
- 快速地将数据送入处理器的途径
  - 从主板芯片组的变迁看如何提高CPU←→Memory传输速率
  - 通过Cache提高CPU←→Memory接口传输速率
  - 总线的串行化：PCI Express
- GPU→GPGPU
  - GPU (Graphics Processing Units)
  - GPGPU (General Purpose computing on Graphics Processing Units)
  - 如何使用GPGPU资源？
- 异构计算(Heterogeneous computing)

谢谢大家