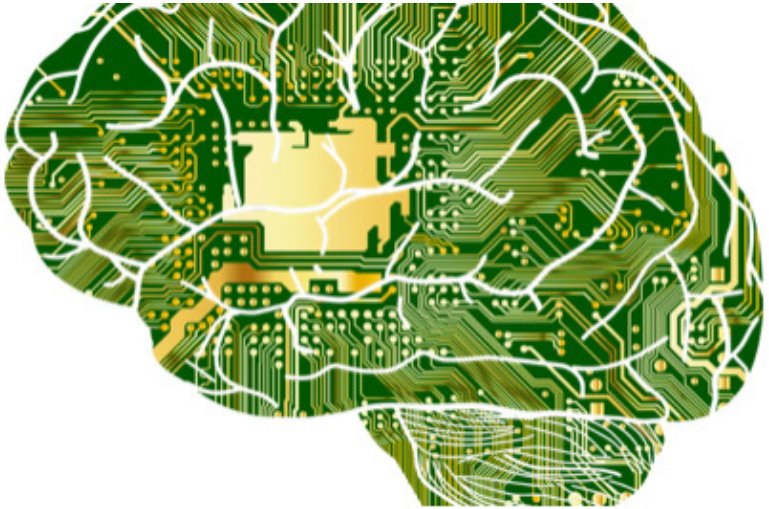
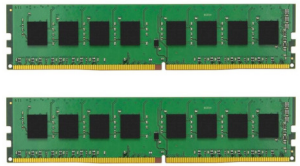
深度学习硬件



你的GPU电脑

Intel i7

0.15 TFLOPS



DDR4

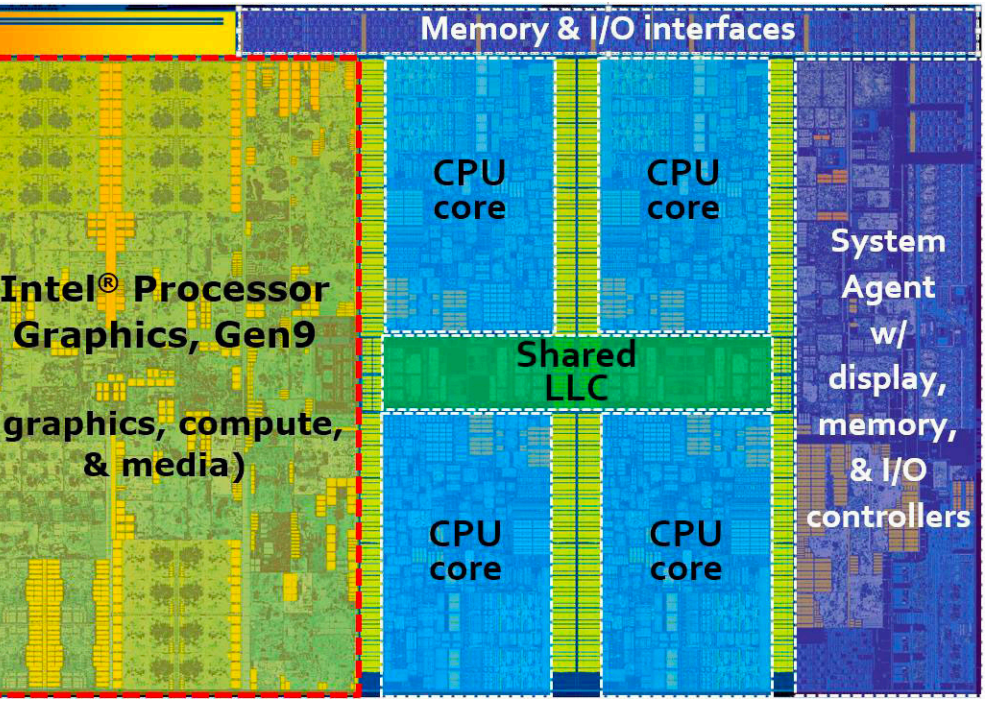
32 GB

Nvidia Titan X

12 TFLOPS

16 GB

Intel i7-6700K

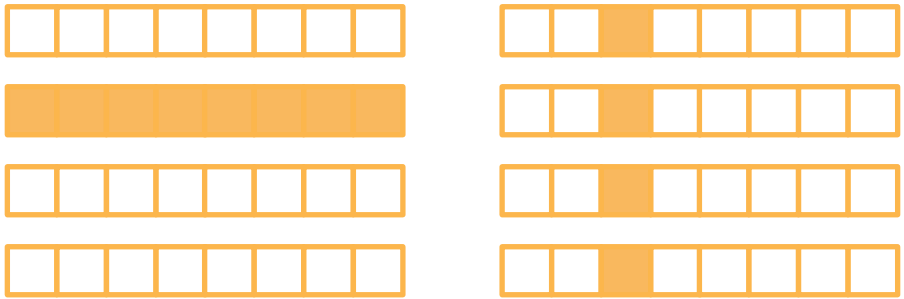


提升CPU利用率 I

- 在计算 $a + b$ 之前，需要准备数据
  - 主内存 -> L3 -> L2 -> L1 -> 寄存器
    - L1 访问延时：0.5 ns
    - L2 访问延时：7 ns (14 x L1)
    - 主内存访问延时：100ns (200 x L1)
- 提升空间和时间的内存本地性
  - 时间：重用数据使得保持它们在缓存里
  - 空间：按序读写数据使得可以预读取

样例分析

- 如果一个矩阵是按列存储，访问一行会比访问一列要快
  - CPU 一次读取 64 字节（缓存线）
  - CPU 会“聪明的”提前读取下一个（缓存线）



提升CPU利用率 II

- 高端 CPU 有几十个核
  - EC2 P3.16xlarge: 2 Intel Xeon CPUs, 32 物理核
- 并行来利用所有核
  - 超线程不一定提升性能，因为它们共享寄存器

样例分析

- 左边比右边慢（python）

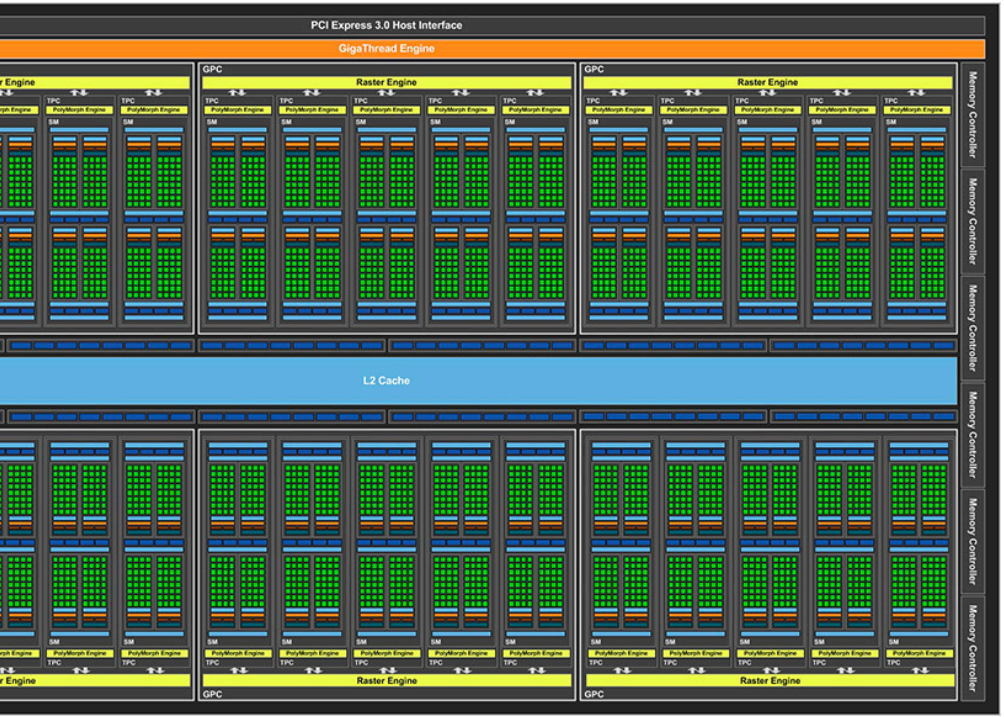
```
for i in range(len(a)):
    c[i] = a[i] + b[i]
```

```
c = a + b
```

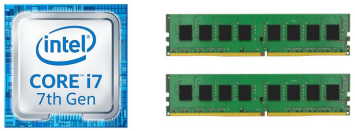
  - 左边调用 n 次函数，每次调用有开销
  - 右边很容易被并行（例如下面C++实现）

```
#pragma omp for
for (i=0; i<a.size(); i++) {
    c[i] = a[i] + b[i]
}
```

Nvidia Titan X (Pascal)



CPU vs GPU



一般 / 高端

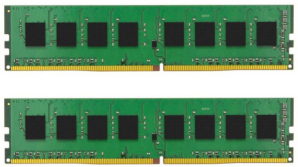
# 核	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
内存大小	32 GB / 1 TB	16 GB / 32 GB
内存带宽	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
控制流	强	弱

提升GPU利用率

- 并行
  - 使用数千个线程
- 内存本地性
  - 缓存更小，架构更加简单
- 少用控制语句
  - 支持有限
  - 同步开销很大

CPU/GPU 带宽

30 GB/s



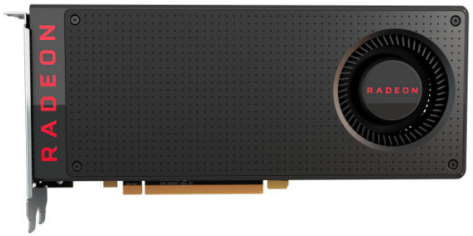
PCIe 3.0 16x: 16 GB/s

480 GB/s

- 不要频繁在CPU和GPU之前传数据：带宽限制，同步开销

更多的 CPUs 和 GPUs

- CPU：AMD，ARM
- GPU：AMD，Intel，ARM，Qualcomm...



CPU/GPU高性能计算编程

- CPU: C++ 或者任何高性能语言
  - 编译器成熟
- GPU
  - Nvidia 上用 CUDA
    - 编译器和驱动成熟
  - 其他用 OpenCL
    - 质量取决于硬件厂商

总结

- CPU：可以处理通用计算。性能优化考虑数据读写效率和多线程
- GPU：使用更多的小核和更好的内存带宽，适合能大规模并行的计算任务