



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

多核程序设计与实践

引言

陶钧

taoj23@mail.sysu.edu.cn

中山大学 数据科学与计算机学院
国家超级计算广州中心

- 课程目标与评分标准
- Why CUDA?
- CUDA加速效果
- 安装与项目创建



- 理解多核程序设计中的重要概念

- 并行与并发，多进程与多线程
 - 同步机制：（进/线程间）通信，原子操作，等

- 掌握一种多核程序设计工具

- CUDA (Compute Unified Device Architecture)

- 能用其将现有算法并行化（课程基本要求）
 - 了解CUDA程序性能优化方法
 - 理解CUDA架构设计原理
 - 了解CUDA程序的运行过程
 - » 线程如何调度，消耗哪些（计算/存储）资源

- 简介其他多核编程框架 (OpenMP, MPI, GPGPU)

- 掌握基本并行模型：归约，扫描，等

● 知识储备需要

- 熟练使用一门编程语言（最好是C或C++）
- 最好了解：计算机图形学、指针与内存、计算机体系结构

● 硬件需要

- 支持CUDA的显卡 (NVIDIA)
 - 往年学院会提供GPU集群供大家使用



- 相关知识背景补充 (1-3周)
 - CUDA与GPGPU、C语言基础、OpenMP
- CUDA与并行计算基础 (4-9周)
 - GPU架构、CUDA线程、内存结构、通信同步机制、性能优化
 - 并行编程模型
 - 其他并行编程框架 (CUDA+MPI, GPGPU与CUDA的关系)
- CUDA应用举例 (剩余)
 - 矩阵乘法, 排序, 图像高斯模糊, Ray Tracing, 等
 - CUDA库选择介绍
 - Thrust, cuDNN, cuBLAS, 等

● 课程总分组成

- 编程作业 50%
- 期中考试 10%
- 期末大作业 30%
- 出勤 10%

● 编程作业评分

- 结果正确性 40%
- 性能 30%
- 尝试通过课程向大家提供更多优秀样板代码?
- 编程规范 10%
- 书面报告 20%

● 编程规范举例

– 提高可读性

- 应组织到适当文件中
- 应根据功能适当划分成函数
- TA应该能轻松读懂你们的代码

文件开头应包含程序说明

```
#####
## 姓名: XXX
## 文件说明: *****
## *****
## *****
## *****
## *****
## *****
#####
#include <abc.h>

int main(){
    ...
}
```

函数开头也应包含说明
程序应适当缩进

```
#####
## 函数: do_something
## 函数描述: *****
## *****
## 参数描述:
## par1: *****
## par2: *****
#####
void do_something(par1, par2){
    for( ... ){
        if( ... ){
            ...
        }
    }
}
```

● 期末大作业

- 完成情况 50%
- 技术难度加分 20%
 - 与完成情况相关联
- 报告 30%
 - 书面报告、现场报告

● 其他

- 允许讨论代码, **严禁抄袭**
- 3 slip days!
 - 应提前通过邮件正式通知TA
 - 包括周末节假日；不适用于大作业的现场报告

- E. Kandrot and J. Sanders, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Pearson, 2010.
 - 中译版：GPU高性能编程：CUDA实战
- N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Prentice Hall, 2013.
 - 中译版：CUDA专家手册：GPU编程权威指南
- J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming*, Wrox, 2014.
 - 中译版：CUDA C编程权威指南
- NVIDIA官方文档：
 - CUDA C Programming Guide
 - CUDA C Programming Best Practice Guide

- 课程目标与评分标准
- Why CUDA?
- CUDA加速效果
- 安装与项目创建



- 摩尔定律

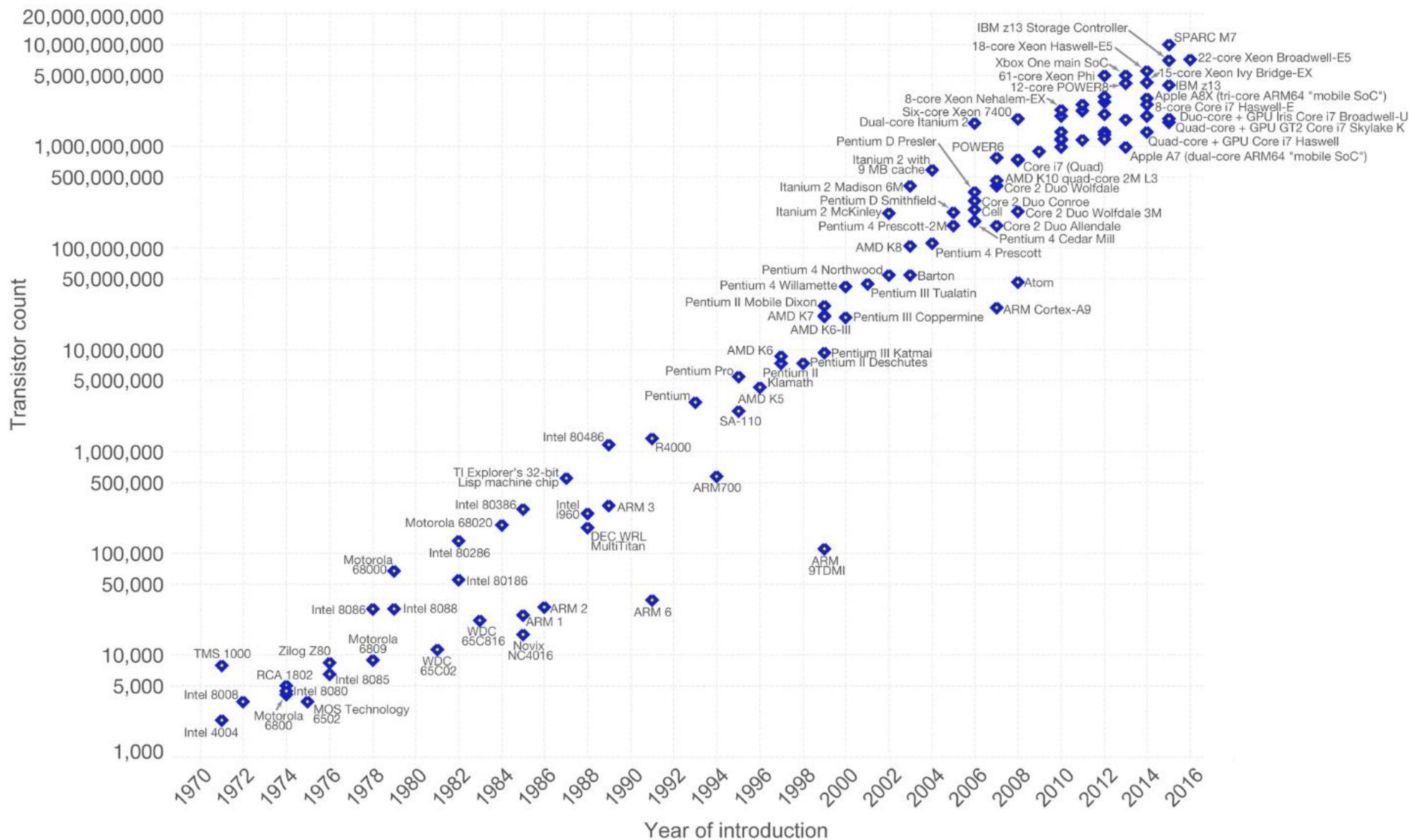
- 英特尔创始人之一戈登·摩尔提出：

“ 集成电路上可容纳的晶体管数目，
约每隔两年便会增长一倍 ”

- 另一版本为18个月（由英特尔首席执行官大卫·豪斯提出）

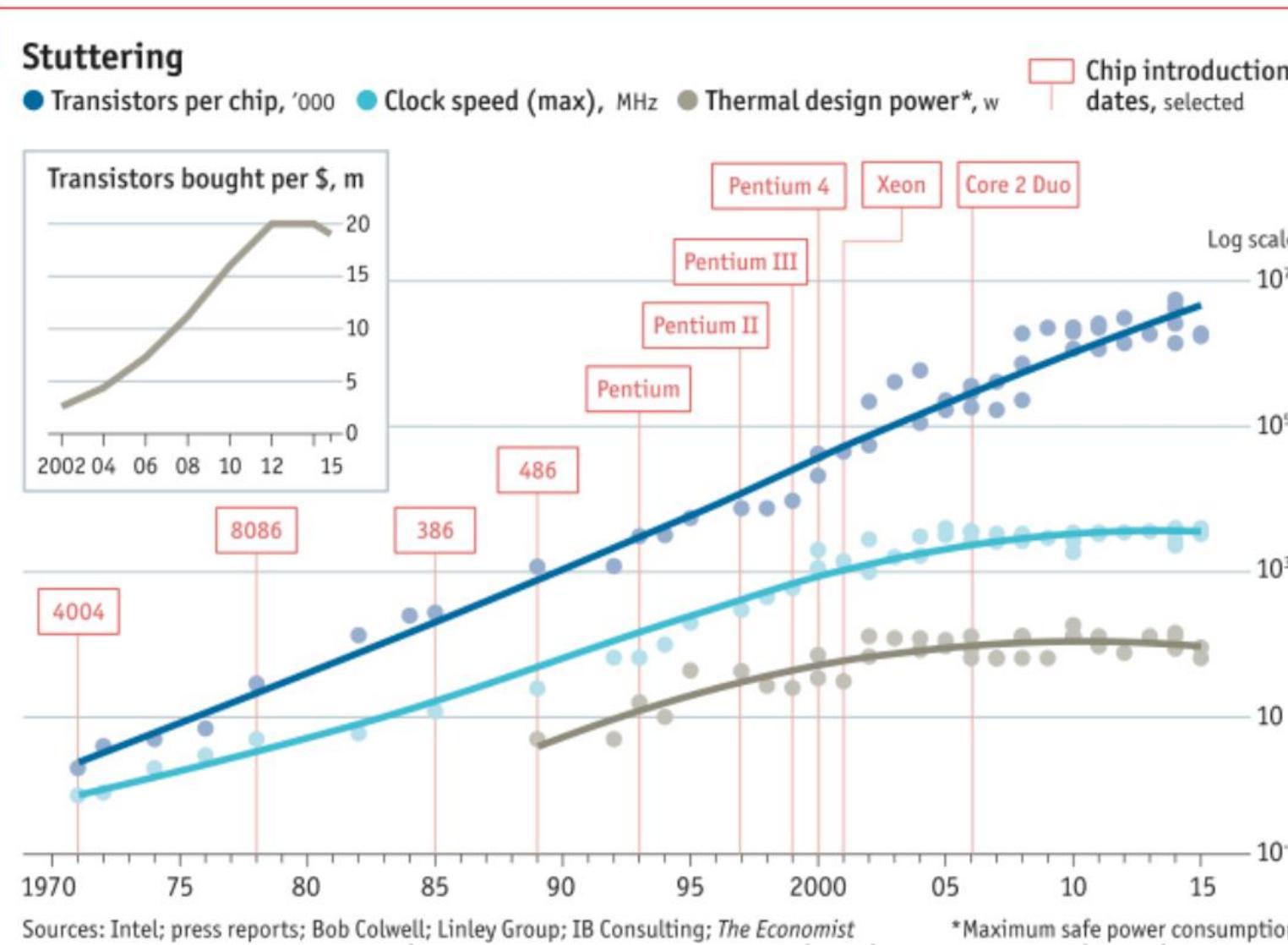


● 摩尔定律



● 摩尔定律的瓶颈

- 晶体管数目 ≠ 频率



摩尔定律还能持续多久？

- 2021/2025年假说
 - 量子穿隧效应

图片来源：The Economist's Technology Quarterly 13

- 串行速度提升已经结束

- 无法继续提升频率
- 无法继续提升功耗

- 当前计算机性能提升趋势

- 计算机没有变得更快，而是变得更宽

- 多核CPU、GPU、超级计算机

- 数据级别的并行

- 同样的指令作用于多个数据

- 线程级别的并行

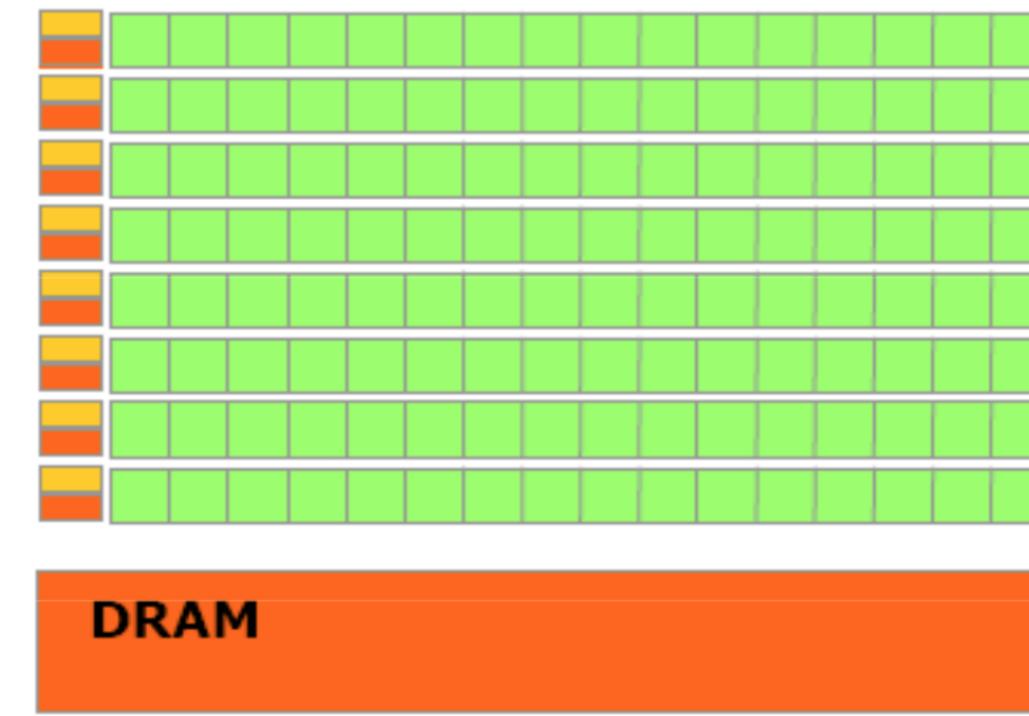
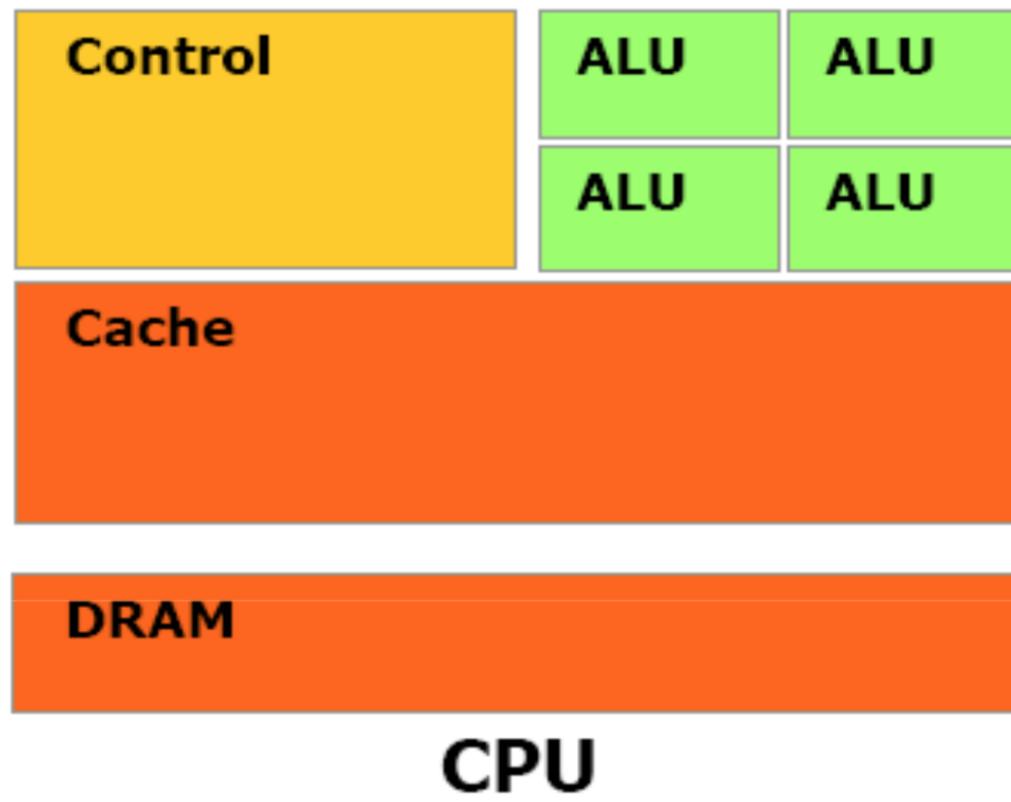


图片来自网络

- GPGPU (General Purpose Computing for GPU)

- GPU特征

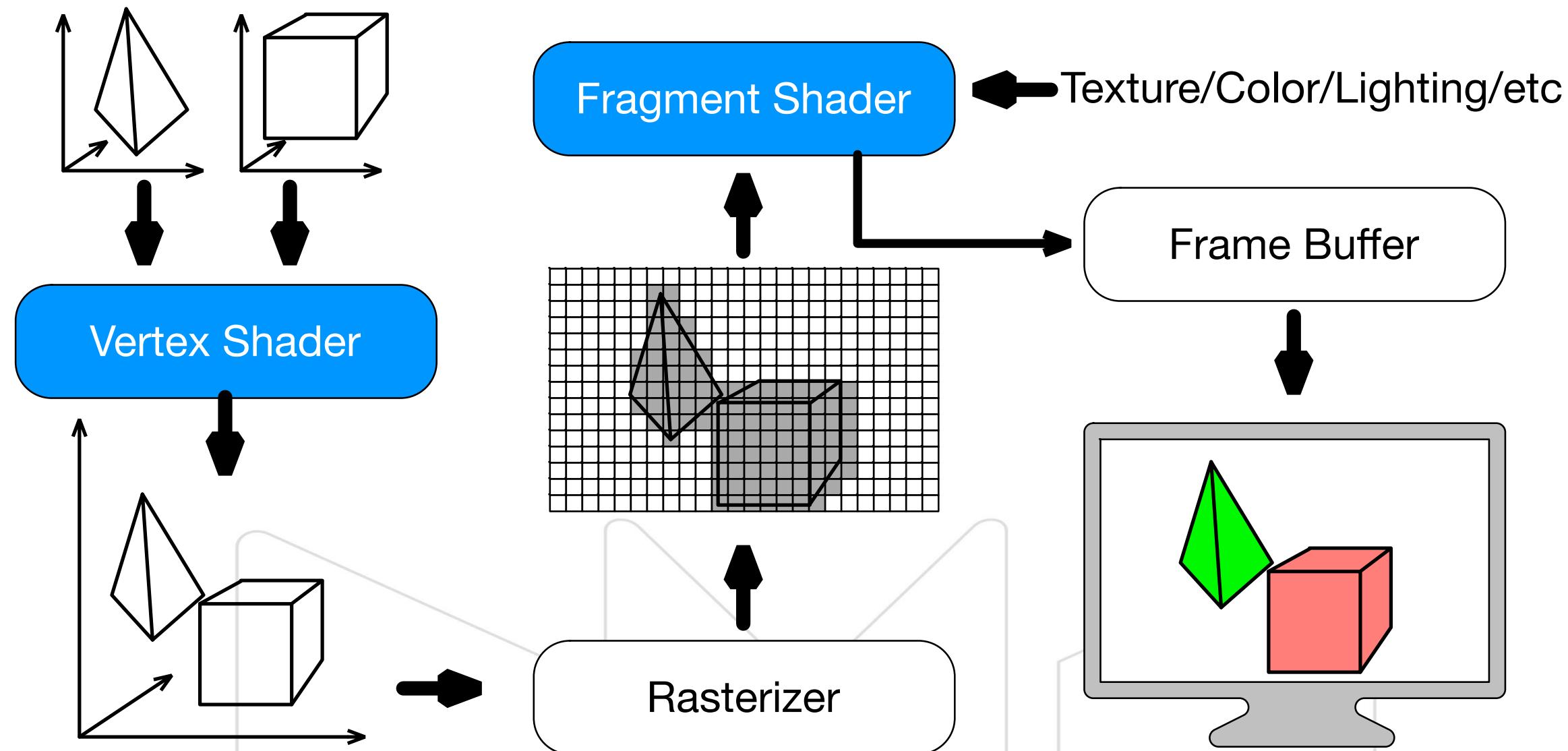
- 高度并行化：同样指令应用于大量数据上
 - 大量运算核心



图片来自NVIDIA

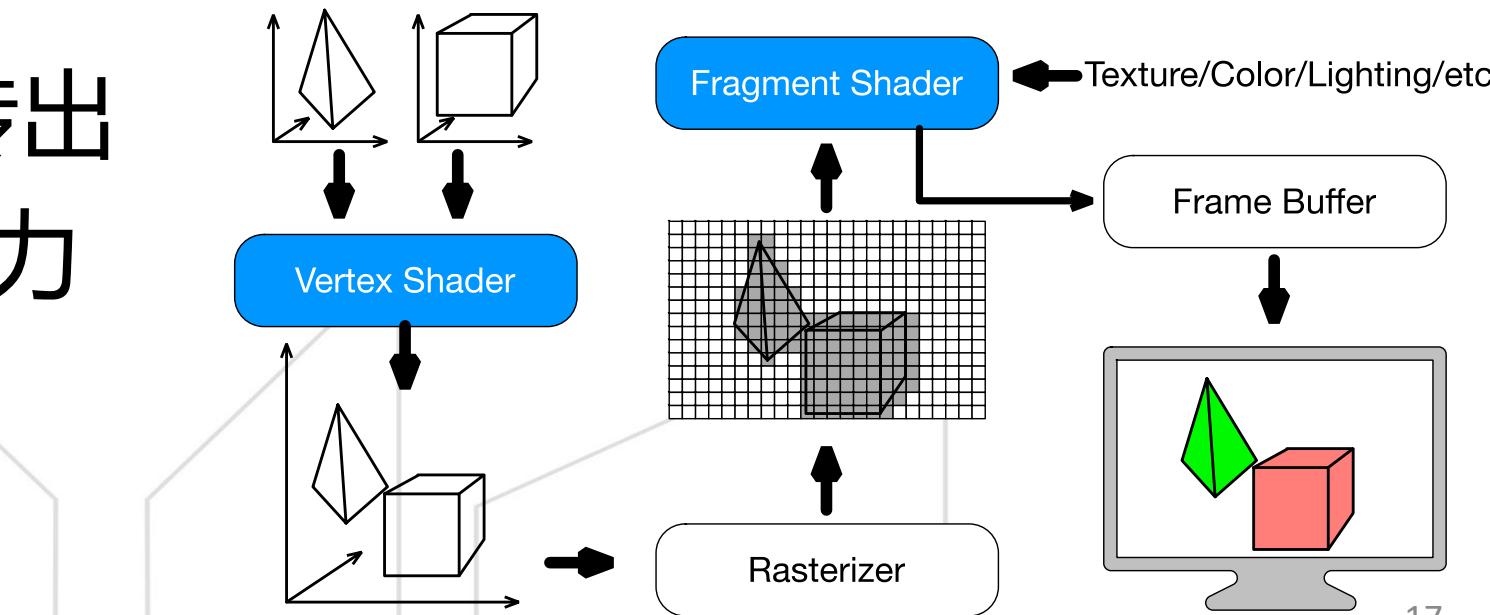
● GPGPU

- 图形绘制简易流程

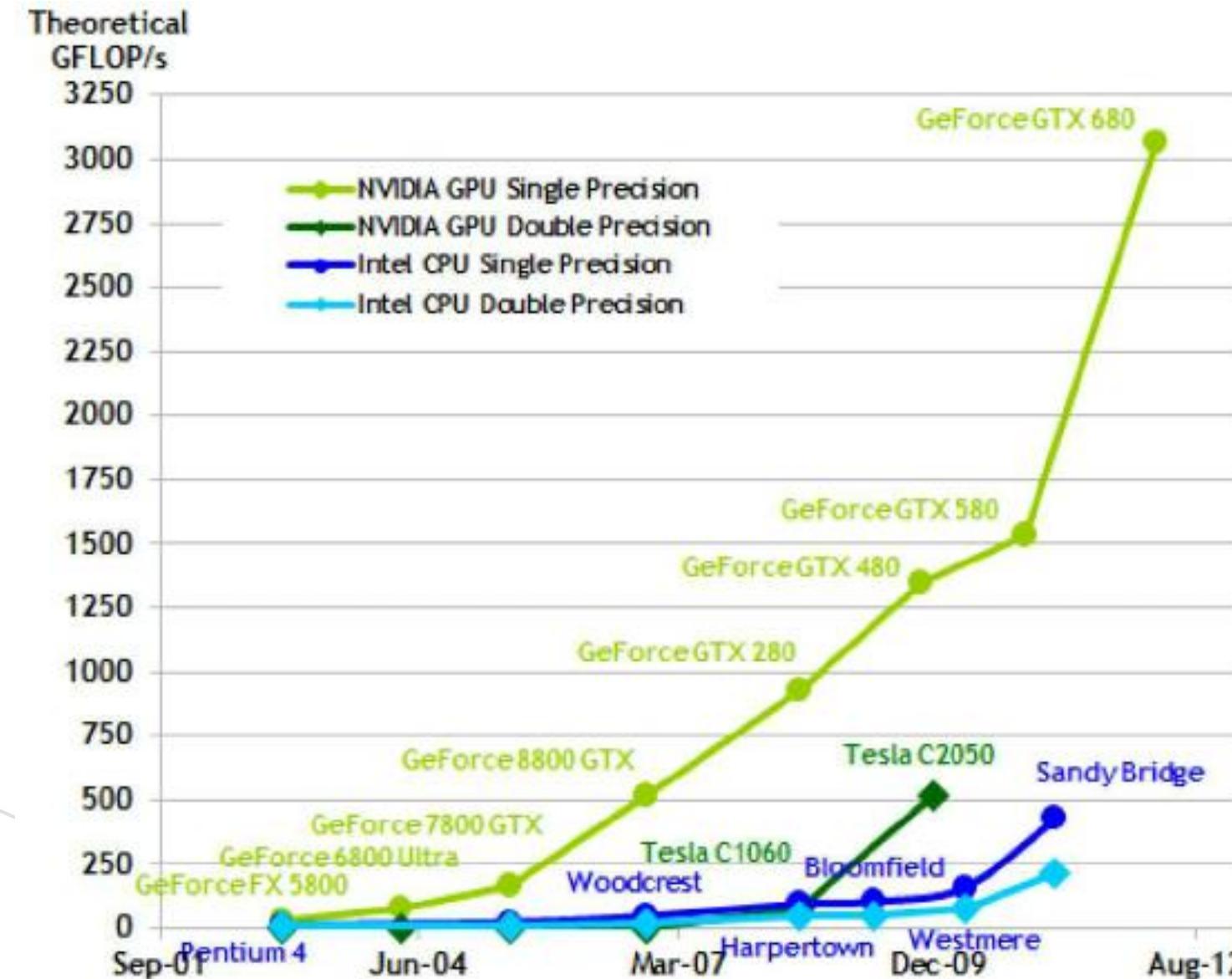


- GPGPU基本思路

- Vertex shader 和 fragment shader 可由用户自行通过 shading language 编程
- 把封装数据封装成图形绘制库所需的数据形式传入
- 绘制过程中通过修改 shader 执行用户编写的程序
 - 通常是 fragment shader
- 将结果绘制到 texture 中传出
- 需要很强的图形学编程能力

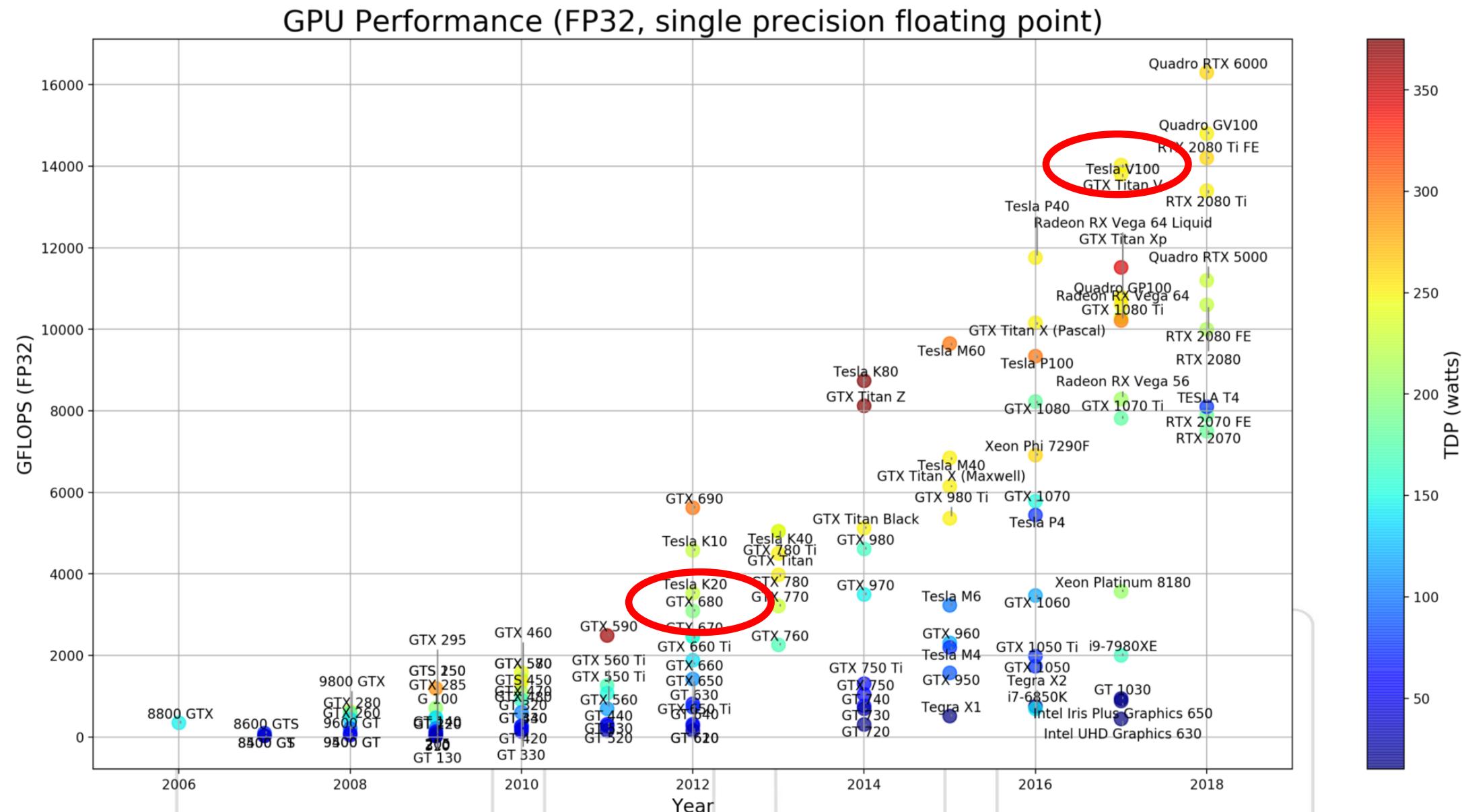


● CPU与GPU性能对比



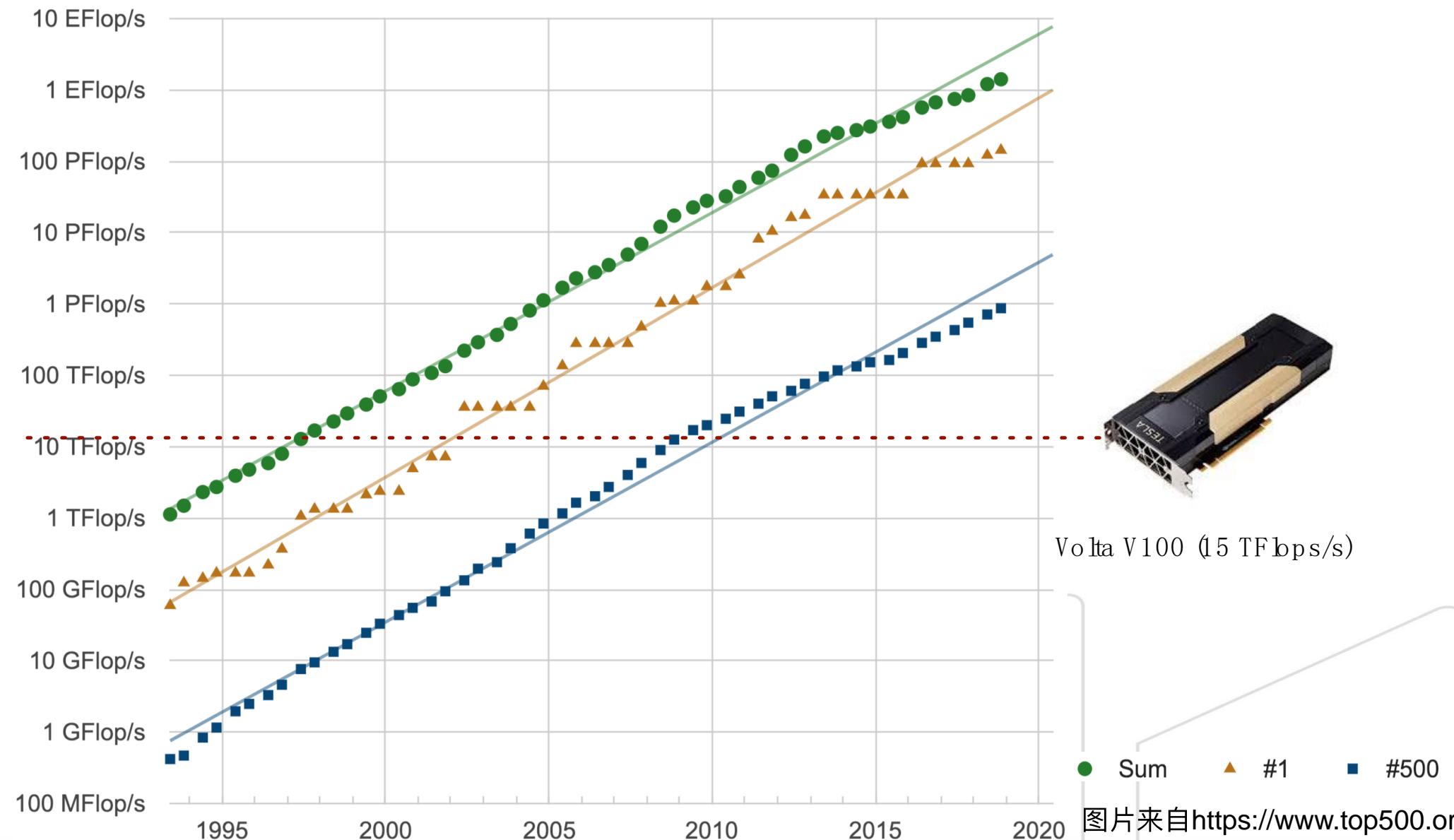
图片来自NVIDIA

● GPU运算速度发展



● GPU与超级计算机

Projected Performance Development



- CUDA vs OpenMP vs MPI

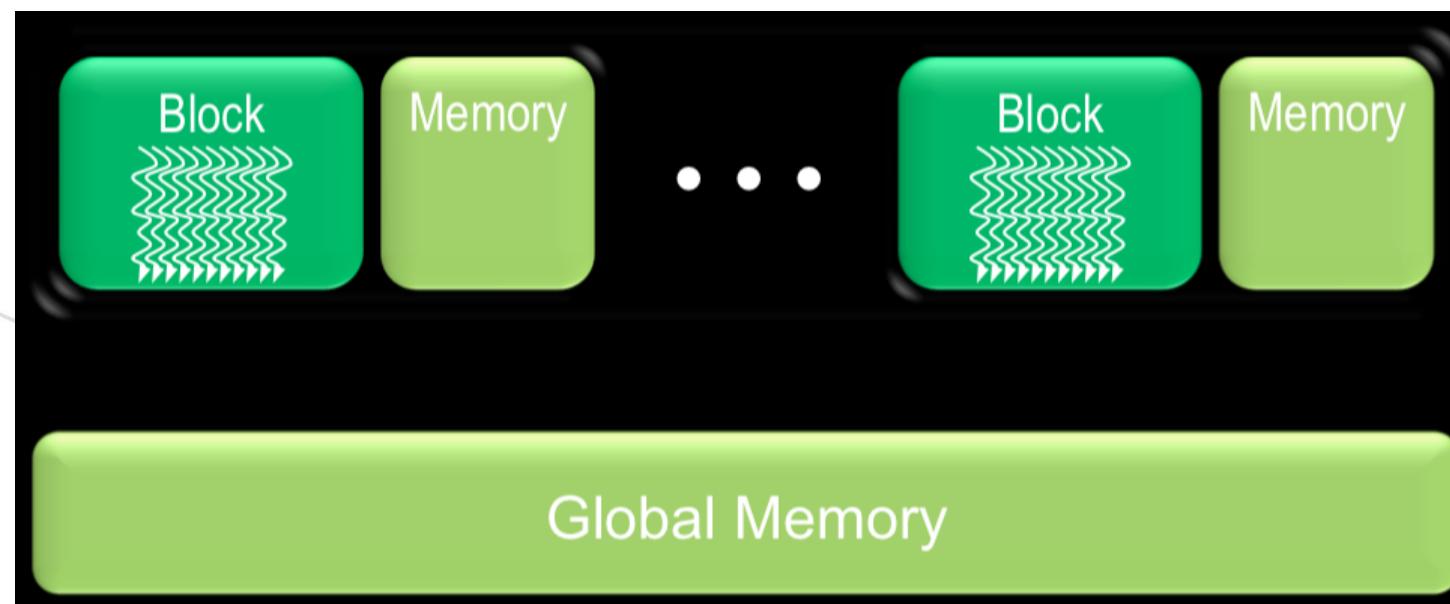
- NVIDIA显卡
 - 基于GPU线程并行



• CUDA vs OpenMP vs MPI

– 硬件结构

- 图中每条波浪线代表一个线程
- 每个block包含多个线程
 - 只有block内部线程能访问其共享内存
- 所有线程均可访问全局存储器

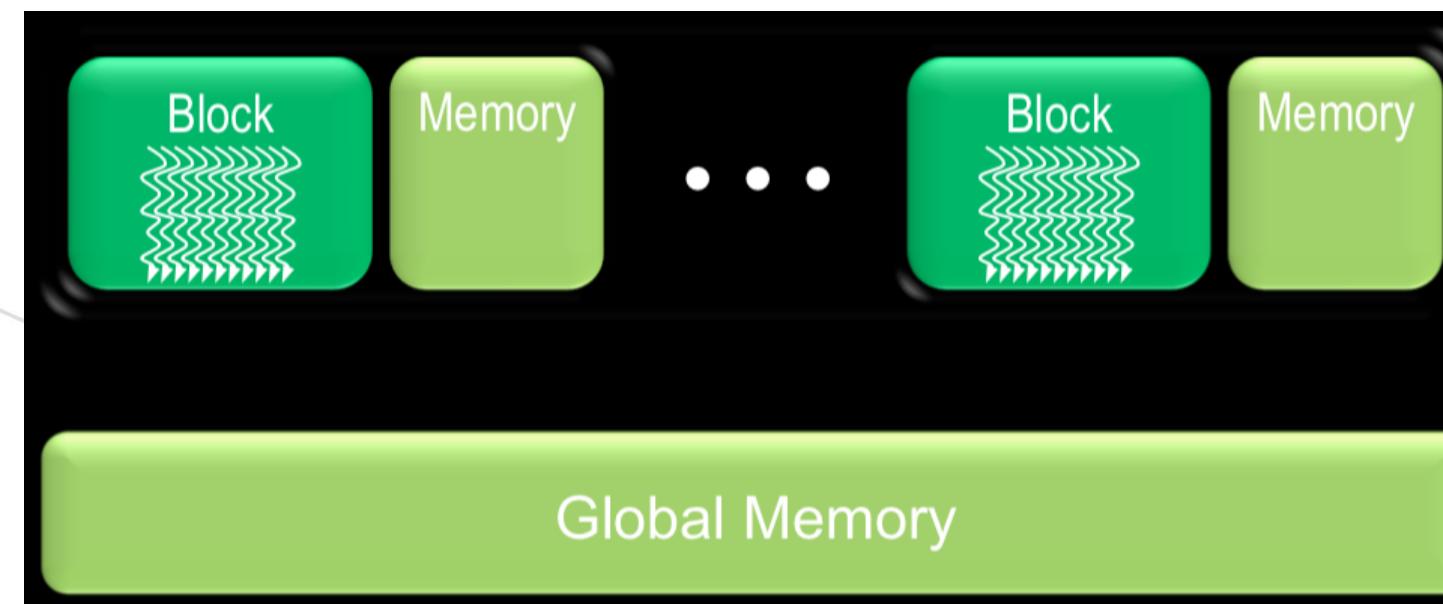


图片来自NVIDIA

• CUDA vs OpenMP vs MPI

– 硬件结构

- 每个线程由一个CUDA core执行
- Block中的线程以warp的形式在streaming multiprocessor上执行
 - 每个warp包含32个thread
 - warp中所有thread执行同样的指令 (Single Instruction Multiple Data)



图片来自NVIDIA

- CUDA vs OpenMP vs MPI

- MP: multiprocessing
- 跨平台
- 基于共享内存的多核系统
 - 多核个人主机
 - 超级计算机 (截止2017年效率并不好)

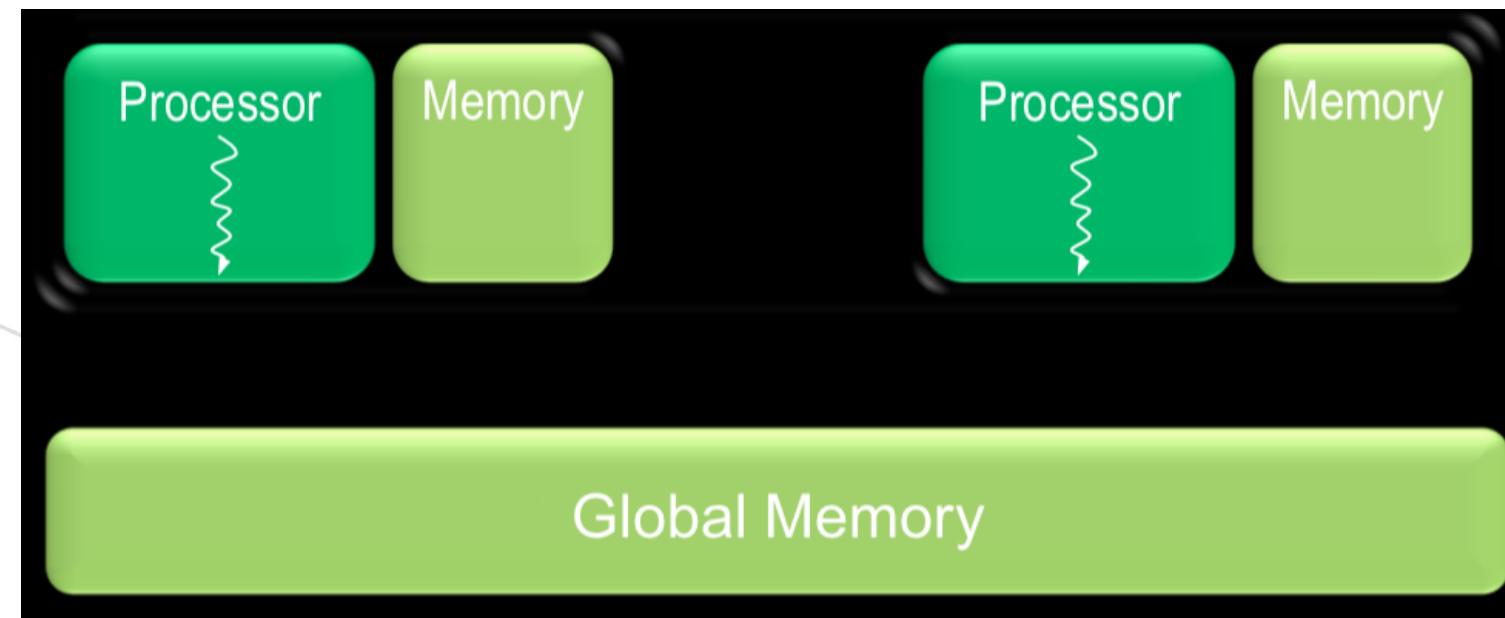


OpenMP®
Enabling HPC since 1997

- CUDA vs OpenMP vs MPI

- 硬件结构

- 每个处理器运行一个线程
 - 每个处理器有易于访问的片上存储 (on-chip memory)
 - 所有处理器共享全局存储器



图片来自NVIDIA

- CUDA vs OpenMP vs MPI
 - MPI (Message Passing Interface)

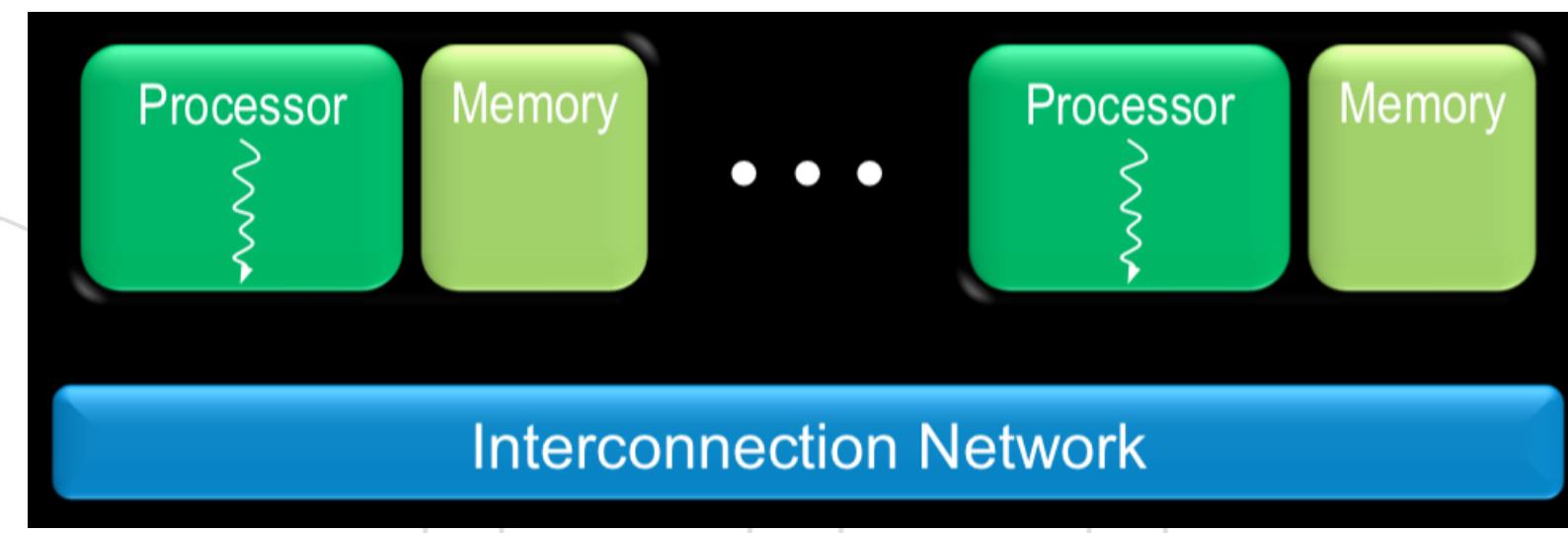
- 跨平台
- 基于信息传递
- 常用于计算机集群



- CUDA vs OpenMP vs MPI

- 硬件结构

- 多个处理器之间通过网络发送消息交互



图片来自NVIDIA

- CUDA vs OpenMP vs MPI

- CUDA

- 创建线程开销低
 - 可创建大量线程
 - 同一warp内共享指令
 - 控制流简单

- OpenMP

- 创建线程开销高
 - 线程数目有限
 - 线程相互间可以完全独立
 - 控制流复杂

- MPI

- 进程相互间可以完全独立
 - 通信成本高
 - 价格昂贵

- CUDA vs OpenCL

- OpenCL: 跨平台的开放标准

- 生态环境:

- CUDA: 用户友好、应用广泛 (目前NVIDIA投入更大)

- OpenCL: 使用稍显复杂、目前应用范围较窄

- 支持硬件:

- CUDA: NVIDIA显卡

- 硬件针对性更强、效率更高、对学习多核编程更有利

- OpenCL: AMD、NVIDIA、Intel、ARM等硬件

- 支持硬件更多、效率因硬件而异、优化因硬件而异

- CUDA vs OpenCL

- OpenCL: 跨平台的开放标准

- 生态环境:

- CUDA: 用户友好、应用广泛 (目前NVIDIA投入更大)
 - OpenCL: 使用稍显复杂、目前应用范围较窄
- "People who are really serious about software should make their own hardware."*

- 支持硬件:

- CUDA: NVIDIA显卡
 - 硬件针对性更强、效率更高、对学习多核编程更有利
 - OpenCL: AMD、NVIDIA、Intel、ARM等硬件
 - 支持硬件更多、效率因硬件而异、优化因硬件而异

-图灵奖得主Alan Kay

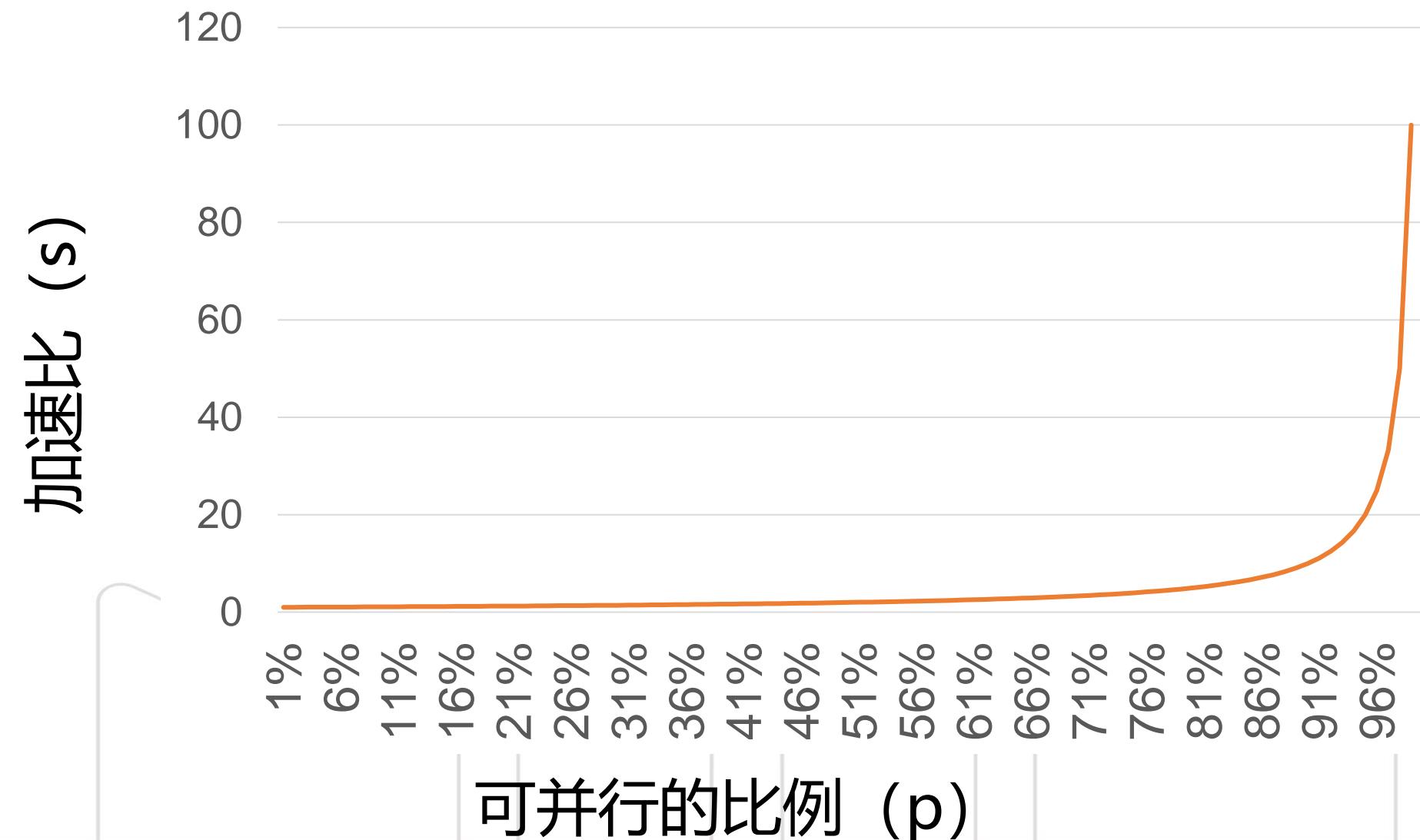
- 课程目标与评分标准
- Why CUDA?
- CUDA加速效果
- 安装与项目创建



● 理论效果 – Amdahl's Law

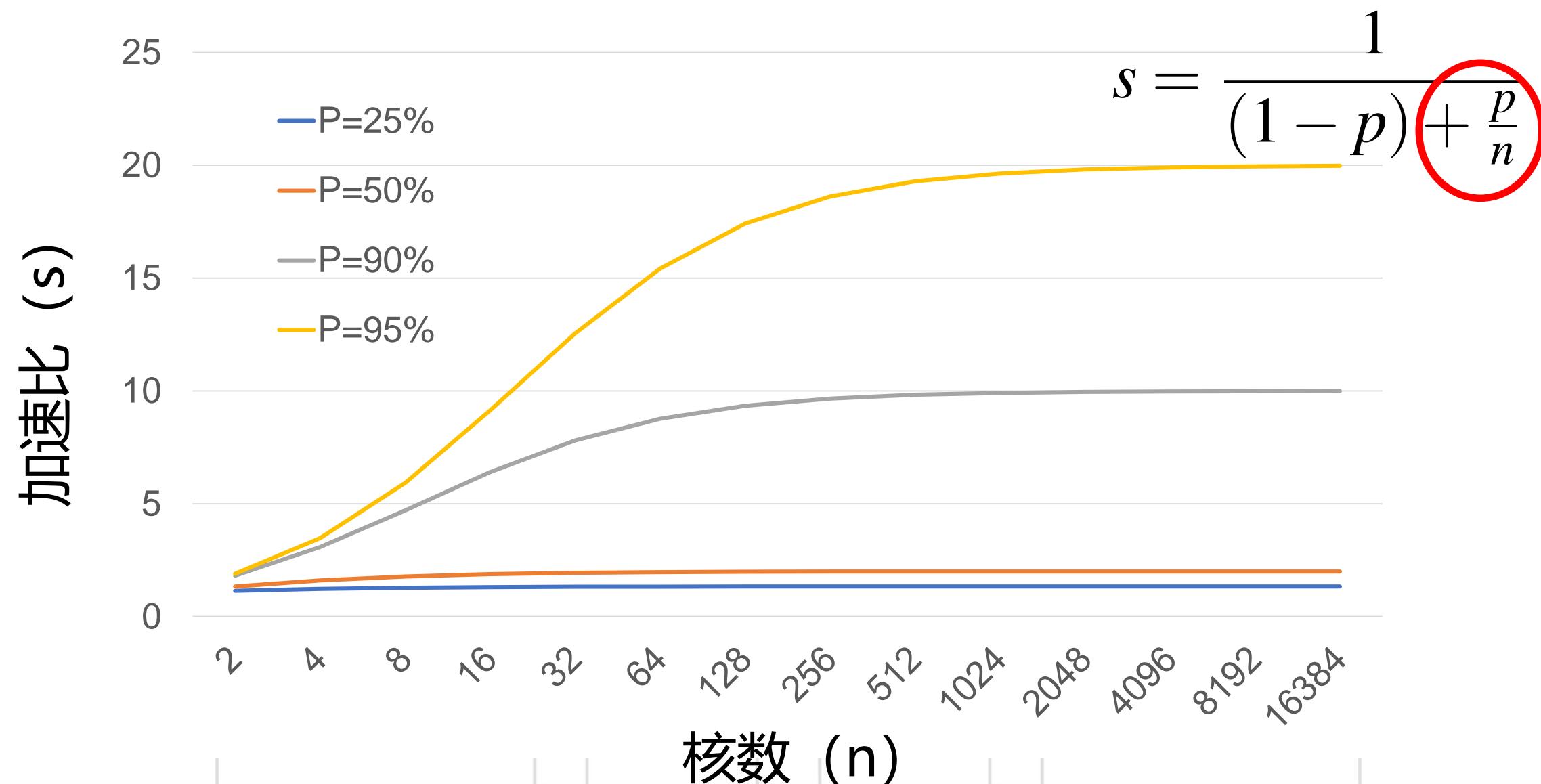
– 程序效率受限于其可并行的比例

$$s = \frac{1}{1 - p}$$



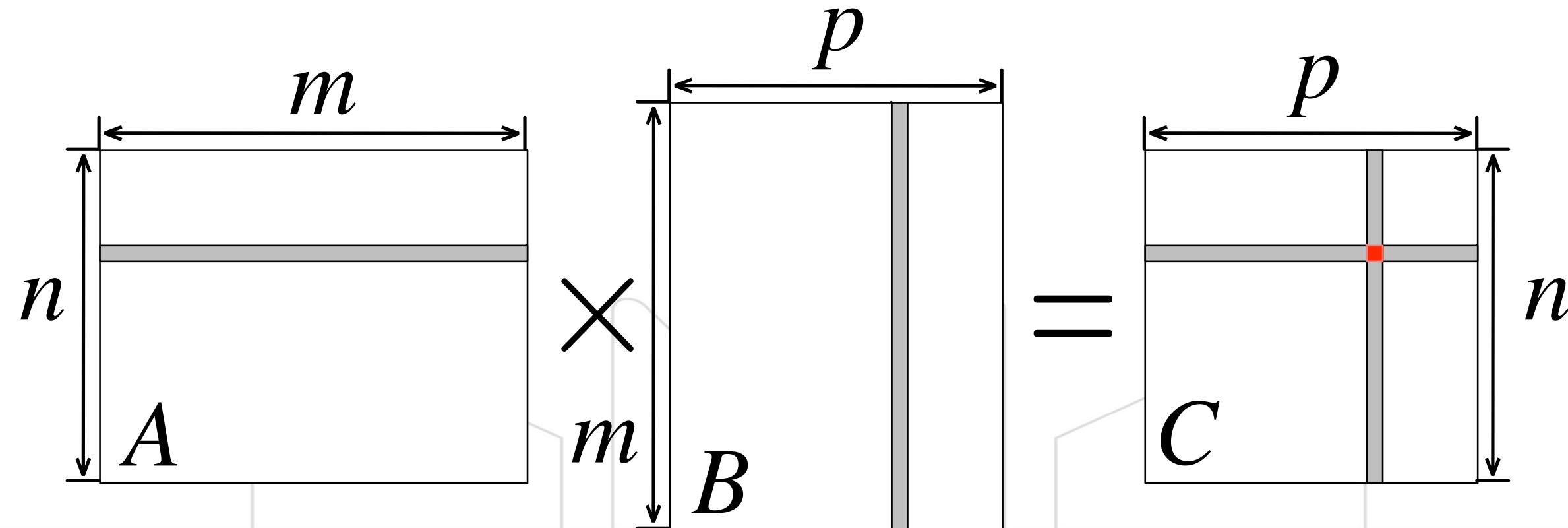
● 理论效果 – Amdahl's Law

– 程序效率受限于其可并行的比例及可供使用的核数



● 案例分析-矩阵乘法

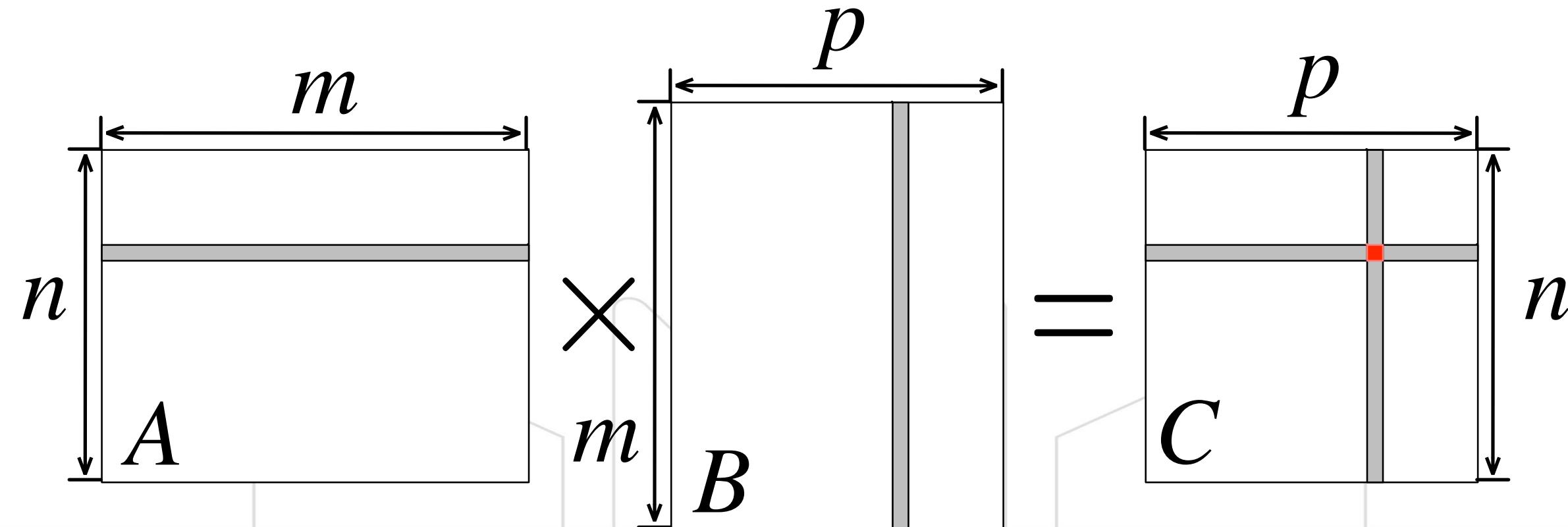
- 对C中所有元素，计算 $C_{ij} = \sum_{k=1}^m A_{ik} \cdot B_{kj}$
 - 共有 $n \times p$ 次向量乘积
 - 每次向量乘积为 m 次乘法与 $m - 1$ 次加法



● 案例分析-矩阵乘法

- 策略1：

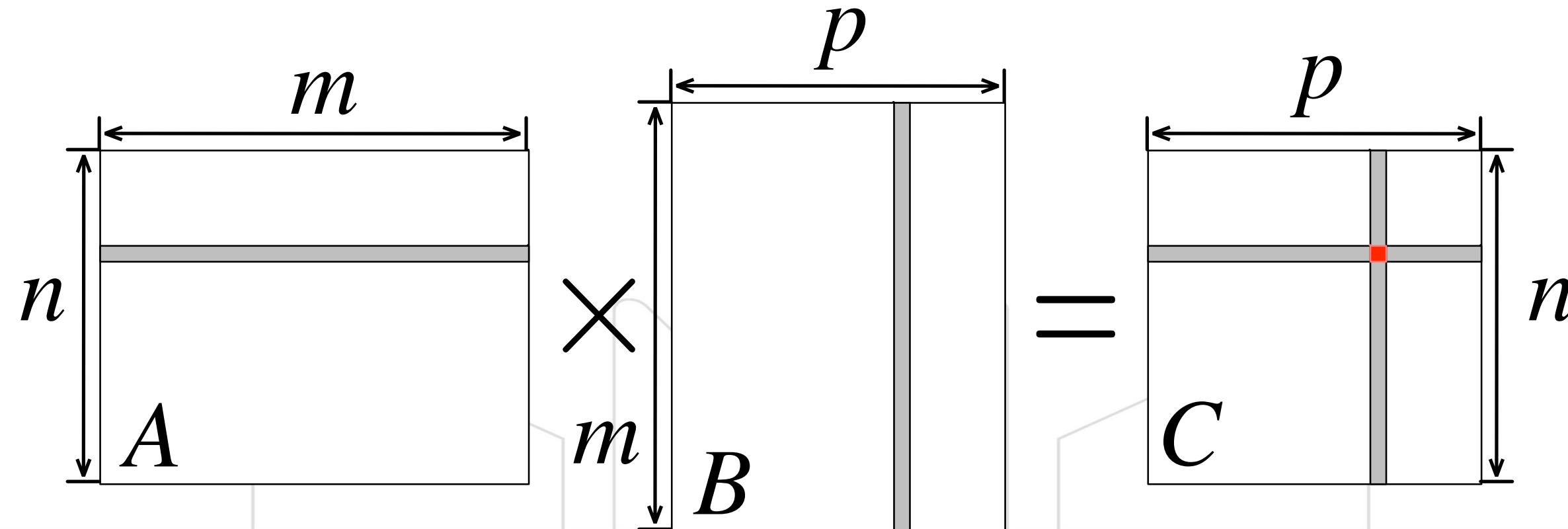
- 创建 $n \times p$ 个线程，每个计算一个向量点积
- 当 $n \times p$ 大于CUDA计算核心数目时，GPU保持满载



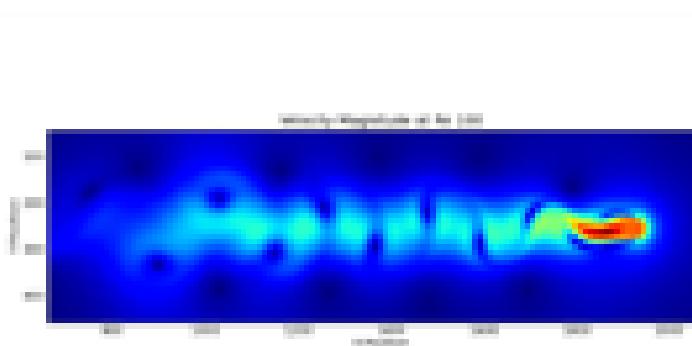
● 案例分析-矩阵乘法

- 策略2：

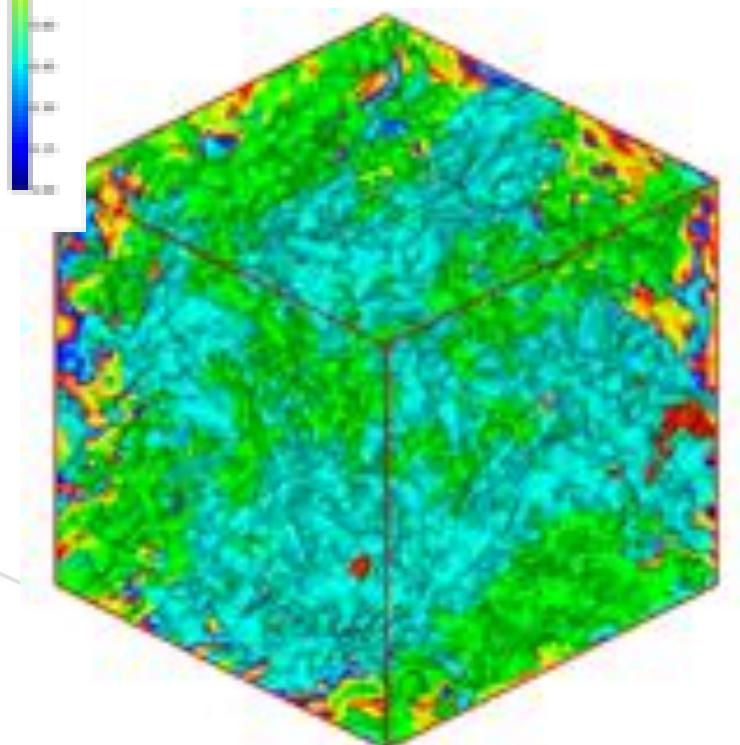
- 先创建 $n \times p \times m$ 个线程计算所有乘法
- 再创建 $n \times p \times (m - 1)$ 个线程计算所有加法
- 更容易保持满载，但创建进程开销也更高



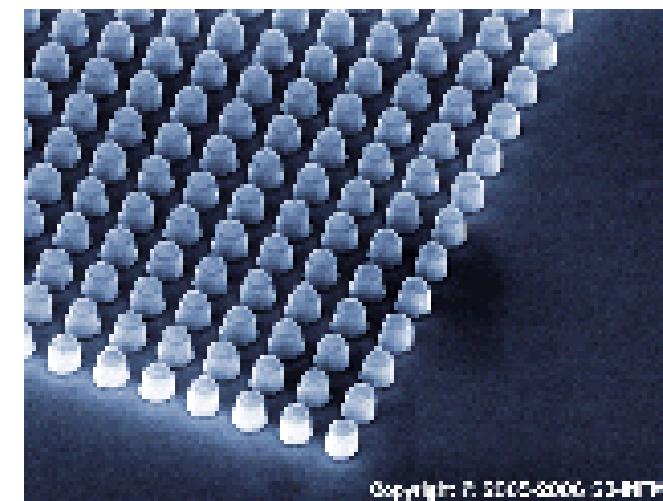
- CUDA Zone Showcase



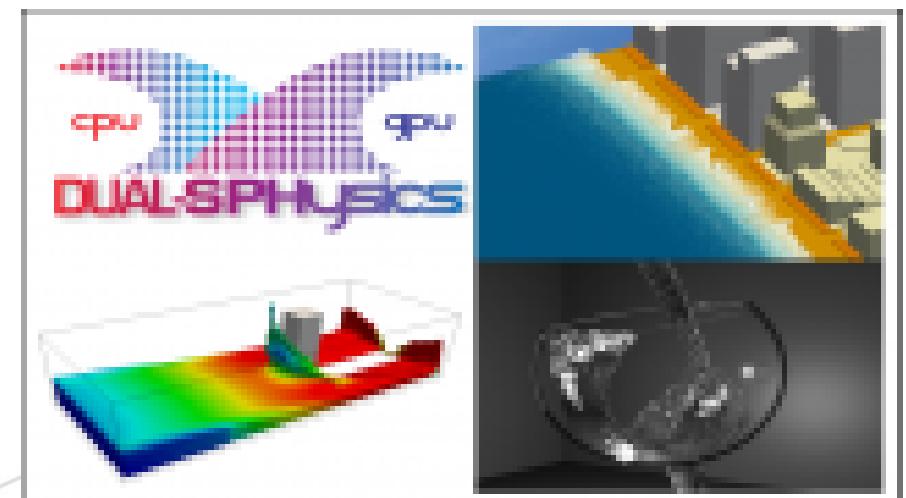
Speedup: 48x



Speedup: 45x

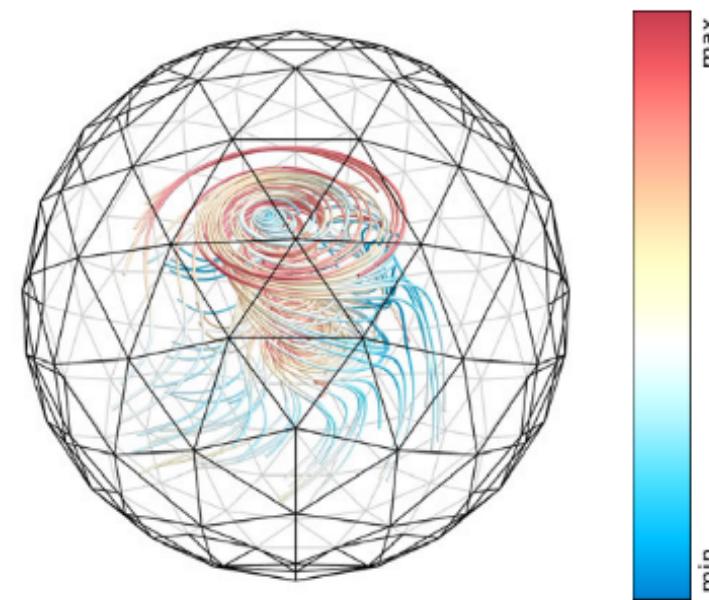
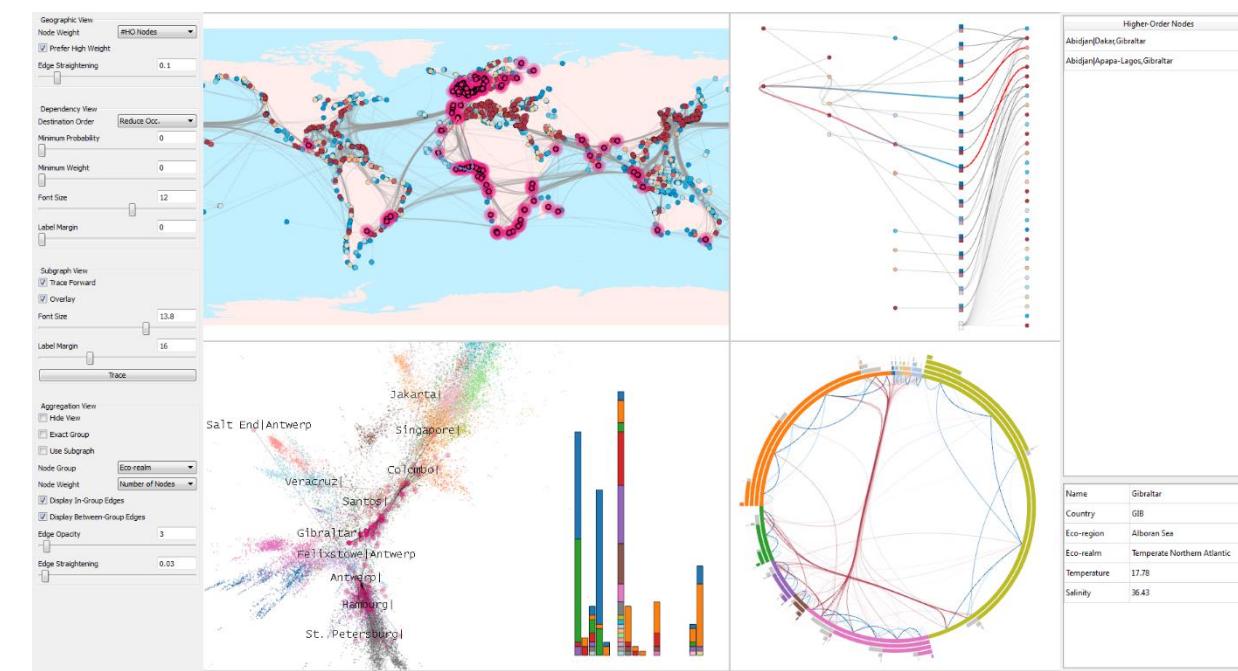
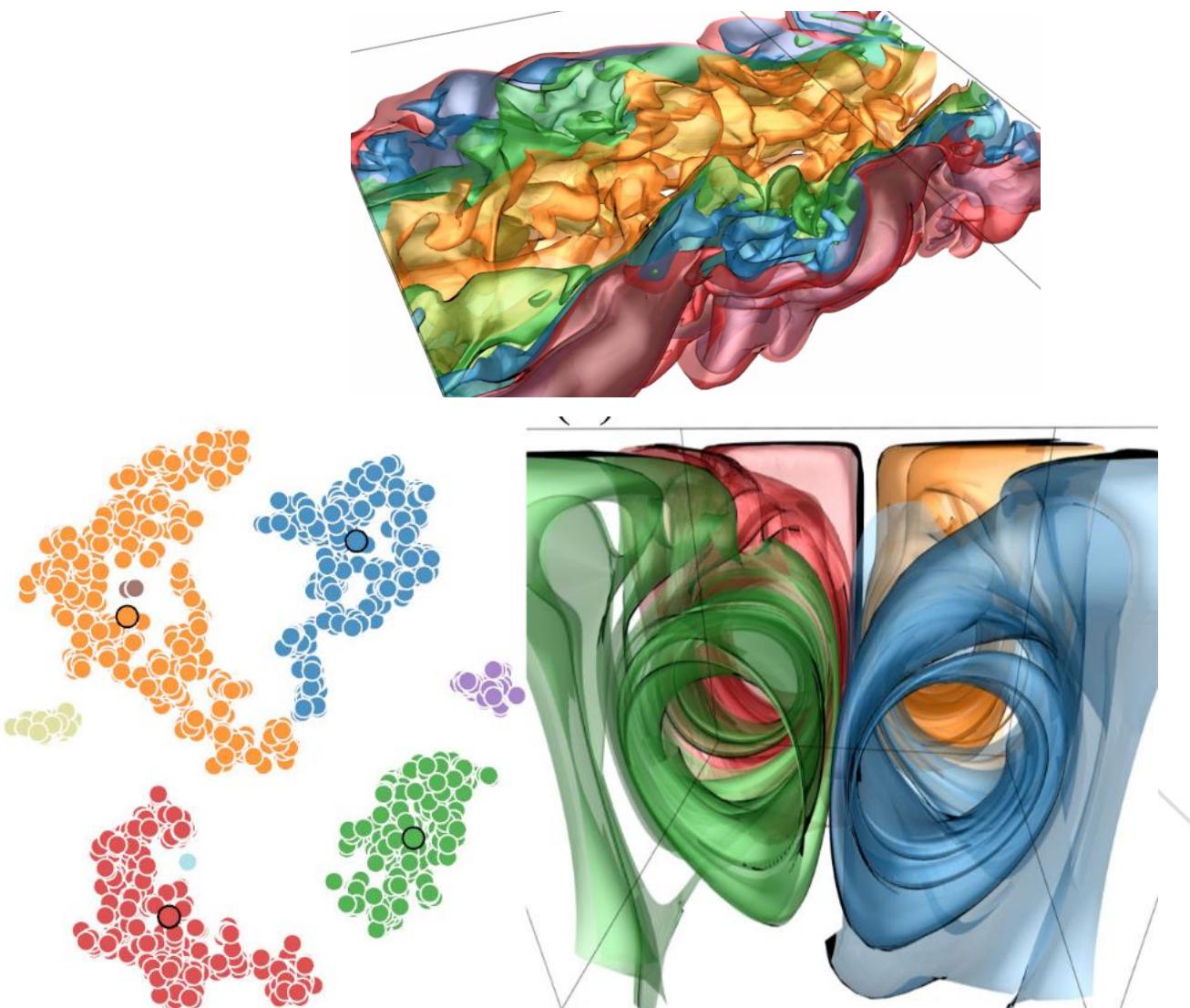


Speedup: 44x



Speedup: 90x

● 我的工作
– 几乎都用到CUDA



$$\begin{matrix} \text{input } p(V) \\ p(v_1) \\ p(v_2) \\ \vdots \\ p(v_m) \end{matrix}$$

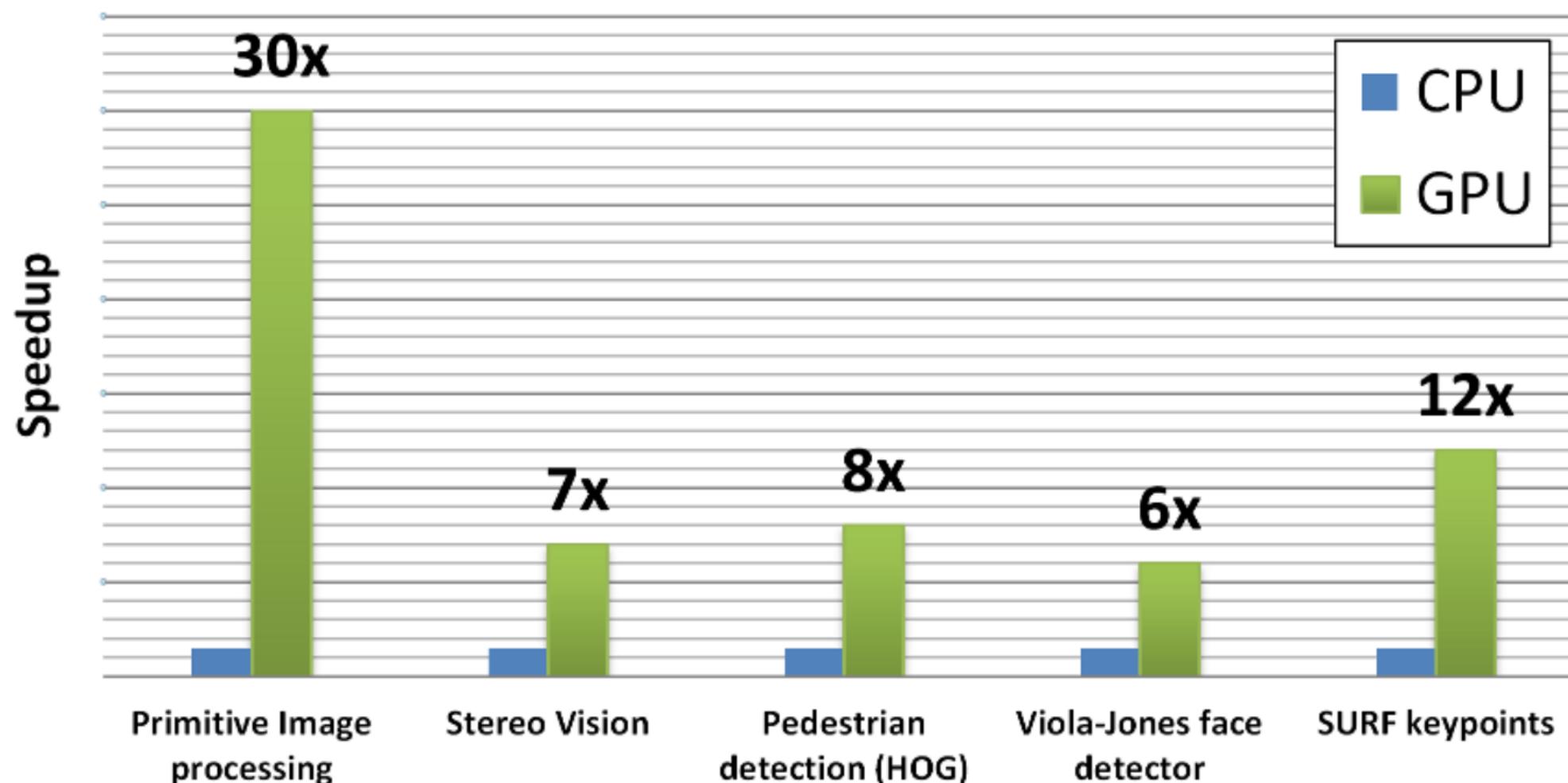
$$\begin{matrix} \text{transition probability matrix } p(S|V) \\ p(s_1|v_1) & p(s_2|v_1) & \dots & p(s_n|v_1) \\ p(s_1|v_2) & p(s_2|v_2) & \dots & p(s_n|v_2) \\ \vdots & \vdots & \ddots & \vdots \\ p(s_1|v_m) & p(s_2|v_m) & \dots & p(s_n|v_m) \end{matrix}$$

$$\begin{matrix} \text{output } p(S) & p(s_1) & p(s_2) & \dots & p(s_n) \end{matrix}$$

information channel $V \rightarrow S$

● OpenCV

– 跨平台计算机视觉库

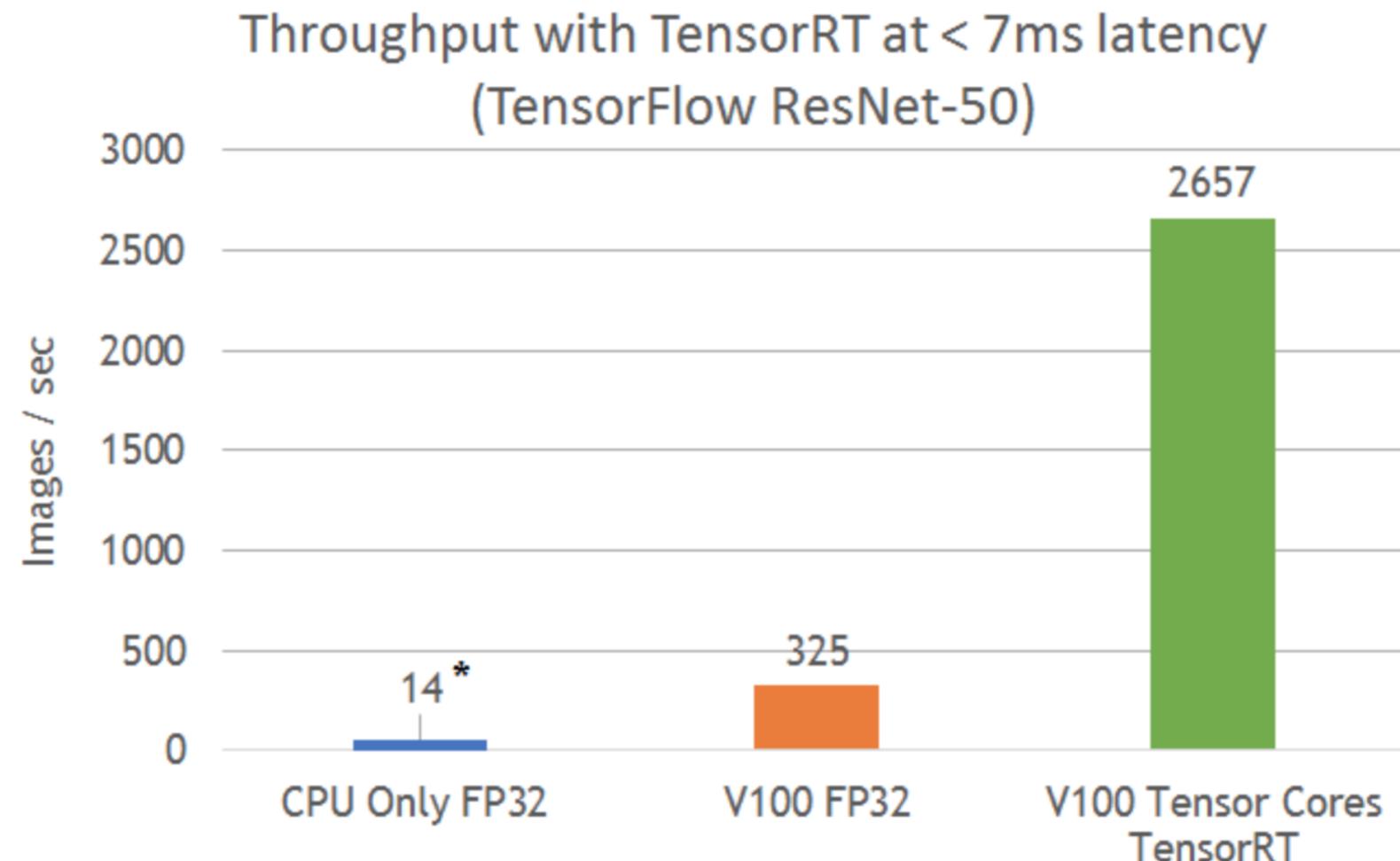


图片来自OpenCV.org

Tesla C2050 (448 cores) versus Core i5-760 2.8Ghz

● TensorFlow

- 深度学习



* Min CPU latency measured was 70 ms. It is not < 7 ms.

CPU: Skylake Gold 6140, 2.5GHz, Ubuntu 16.04; 18 CPU threads. Volta V100 SXM; CUDA (384.111; v9.0.176);

Batch sizes: CPU=1, V100_FP32=2, V100_TensorFlow_TensorRT=16 w/ latency=6ms

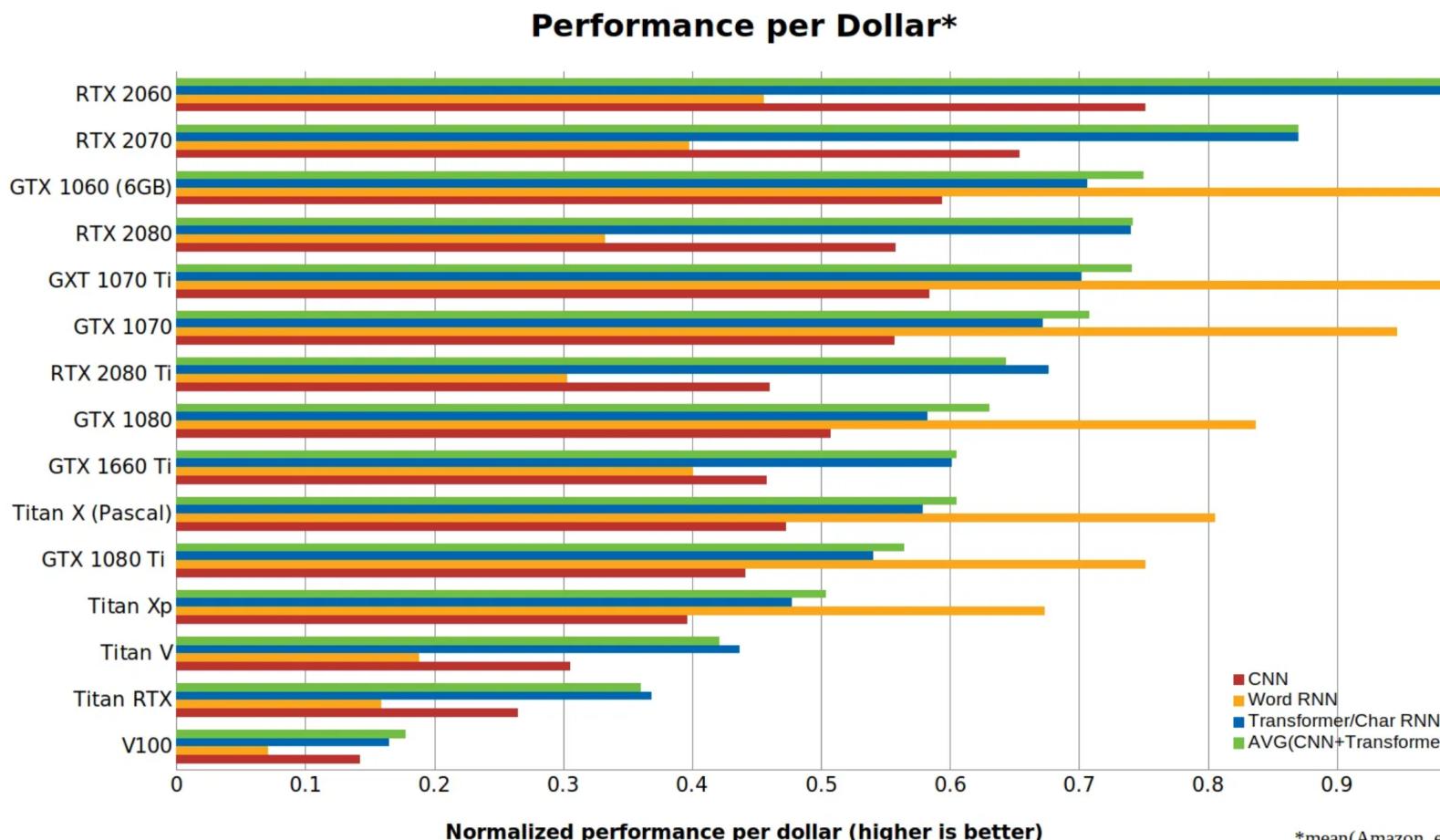
图片来自 Laurence Moroney

● 选择你的硬件

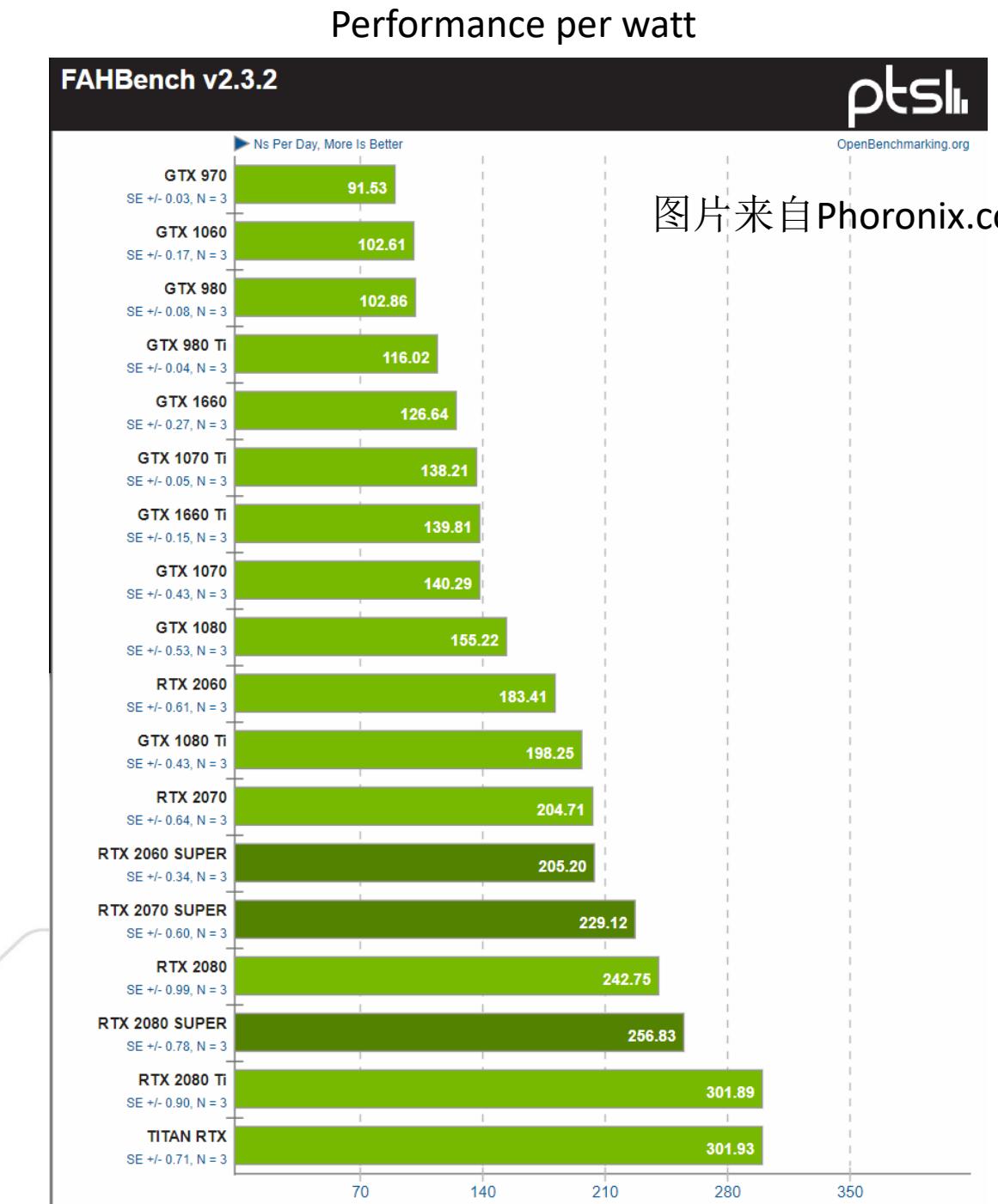
– 显存，半精度/单精度/双精度运算能力

GPU	FP64	FP16	Memory
GeForce GTX 1080 Ti	up to 0.355 TFLOPS	less than 0.177 TFLOPS	11GB
GeForce Titan Xp	up to 0.380 TFLOPS	less than 0.190 TFLOPS	12GB
GeForce Titan V	up to 6.875 TFLOPS	~27.5 TFLOPS	12GB
GeForce RTX 2080 Ti	estimated ~0.44 TFLOPS	28.5 TFLOPS	11GB
Titan RTX	estimated ~0.51 TFLOPS	up to 32.6 TFLOPS	24GB
Tesla K80	1.87+ TFLOPS	N/A	24GB
Tesla P100	4.7 ~ 5.3 TFLOPS	18.7 ~ 21.2 TFLOPS	12GB or 16GB
Quadro GP100	5.2 TFLOPS	20.7 TFLOPS	16GB
Tesla V100	7 ~ 7.8 TFLOPS	28 ~ 31.4 TFLOPS	16GB or 32GB
Quadro GV100	7.4 TFLOPS	29.6 TFLOPS	32GB
Quadro RTX 6000 and 8000	~ 0.5 TFLOPS	32.6 TFLOPS	24/48GB
Tesla T4	estimated ~0.25 TFLOPS	16.2 TFLOPS	16GB

● 选择你的硬件
— 性价比

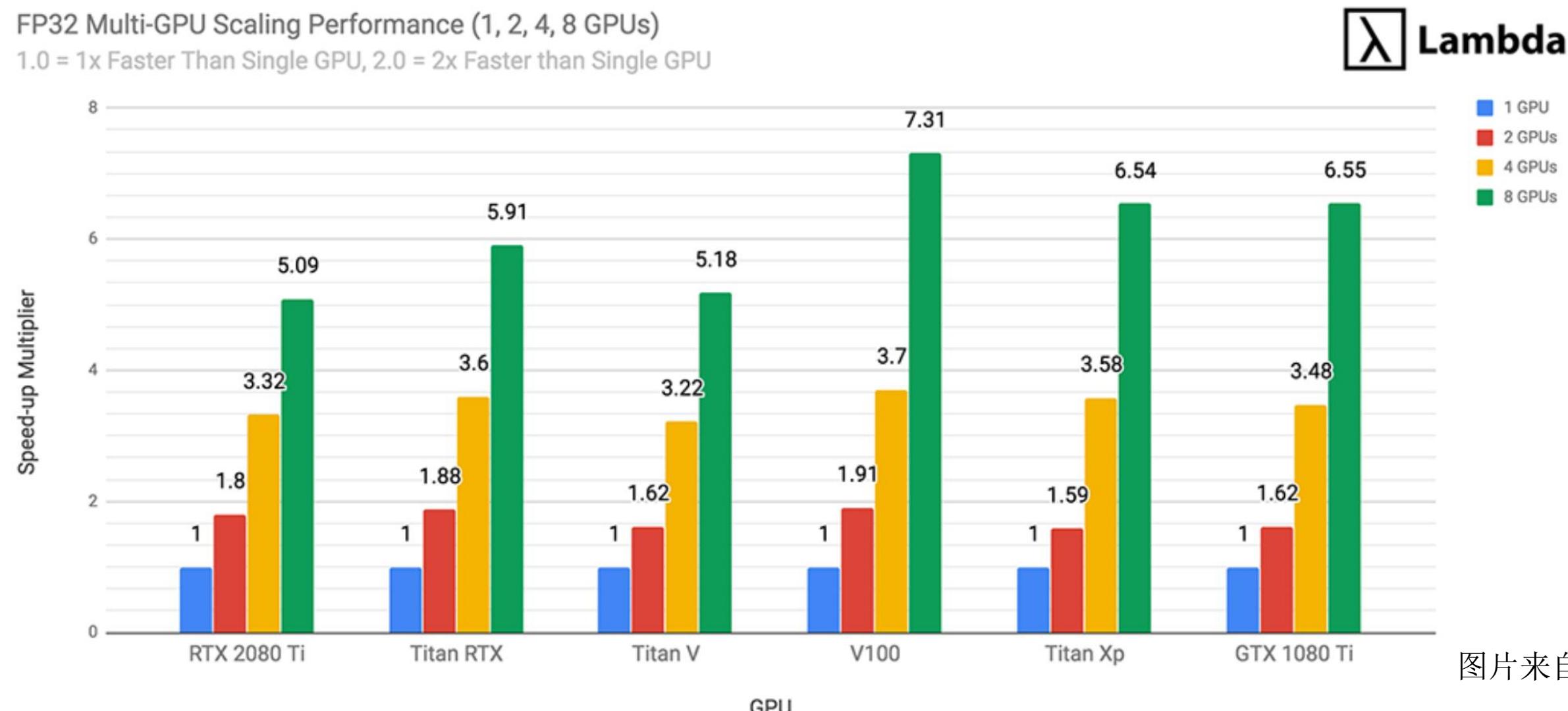


图片来自Tim Dettmers



- 选择你的硬件
 - 多GPU并行能力

FP32 Multi-GPU Scaling Performance (1, 2, 4, 8 GPUs)
1.0 = 1x Faster Than Single GPU, 2.0 = 2x Faster than Single GPU



图片来自LambdaLabs

- 课程目标与评分标准
- Why CUDA?
- CUDA加速效果
- 安装与项目创建



● 下载CUDA Toolkit

- 下载地址: <http://developer.nvidia.com/cuda-downloads>
- Toolkit
 - nvcc 编译器
 - visual profiler
 - visual studio integration (windows)
- Samples
 - 基本功能: device query、band width query等
 - 线性系统: QR factorization, LU decomposition等
 - 图算法: 最短路径、PageRank等
 - 模拟: 粒子系统、ray tracing等
- 显卡驱动

● 安裝

– 參照說明文檔

- <https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>
- <https://docs.nvidia.com/cuda/cuda-installation-guide-mac-os-x/index.html>
- <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html>

安裝成功後應該能通過
deviceQuery查看支持
CUDA的硬件狀態

```
deviceQuery.exe Starting...
CUDA Device Query <Runtime API> version <CUDART static linking>
Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 680"
  CUDA Driver Version / Runtime Version      6.0 / 6.0
  CUDA Capability Major/Minor version number: 3.0
  Total amount of global memory:             2048 MBytes (2147483648 bytes)
  < 8> Multiprocessors, <192> CUDA Cores/MP:
    GPU Clock rate:                         1059 MHz (1.06 GHz)
    Memory Clock rate:                      3004 Mhz
    Memory Bus Width:                       256-bit
    L2 Cache Size:                          524288 bytes
    Maximum Texture Dimension Size <x,y,z>: 1D=<65536>, 2D=<65536, 65536>, 3D=<4096, 4096, 4096>
    Maximum Layered 1D Texture Size, <num> layers: 1D=<16384>, 2048 layers
    Maximum Layered 2D Texture Size, <num> layers: 2D=<16384, 16384>, 2048 layers
    Total amount of constant memory:          65536 bytes
    Total amount of shared memory per block:  49152 bytes
    Total number of registers available per block: 65536
    Warp size:                             32
    Maximum number of threads per multiprocessor: 2048
    Maximum number of threads per block:       1024
    Max dimension size of a thread block <x,y,z>: <1024, 1024, 64>
    Max dimension size of a grid size <x,y,z>: <2147483647, 65535, 65535>
    Maximum memory pitch:                   2147483647 bytes
    Texture alignment:                     512 bytes
    Concurrent copy and kernel execution: Yes with 1 copy engine(s)
    Run time limit on kernels:            Yes
    Integrated GPU sharing Host Memory: No
    Support host page-locked memory mapping: Yes
    Alignment requirement for Surfaces: Yes
    Device has ECC support:               Disabled
    CUDA Device Driver Mode (TCC or WDDM): WDDM <Windows Display Driver Model>
    Device supports Unified Addressing (UVA): No
    Device PCI Bus ID / PCI location ID: 1 / 0
    Compute Mode:
      < Default <multiple host threads can use ::cudaSetDevice() with device simultaneously> >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.0, CUDA Runtime Version = 6.0, NumDevs = 1, Device0 = GeForce GTX 680
Result = PASS
```

● 简单程序

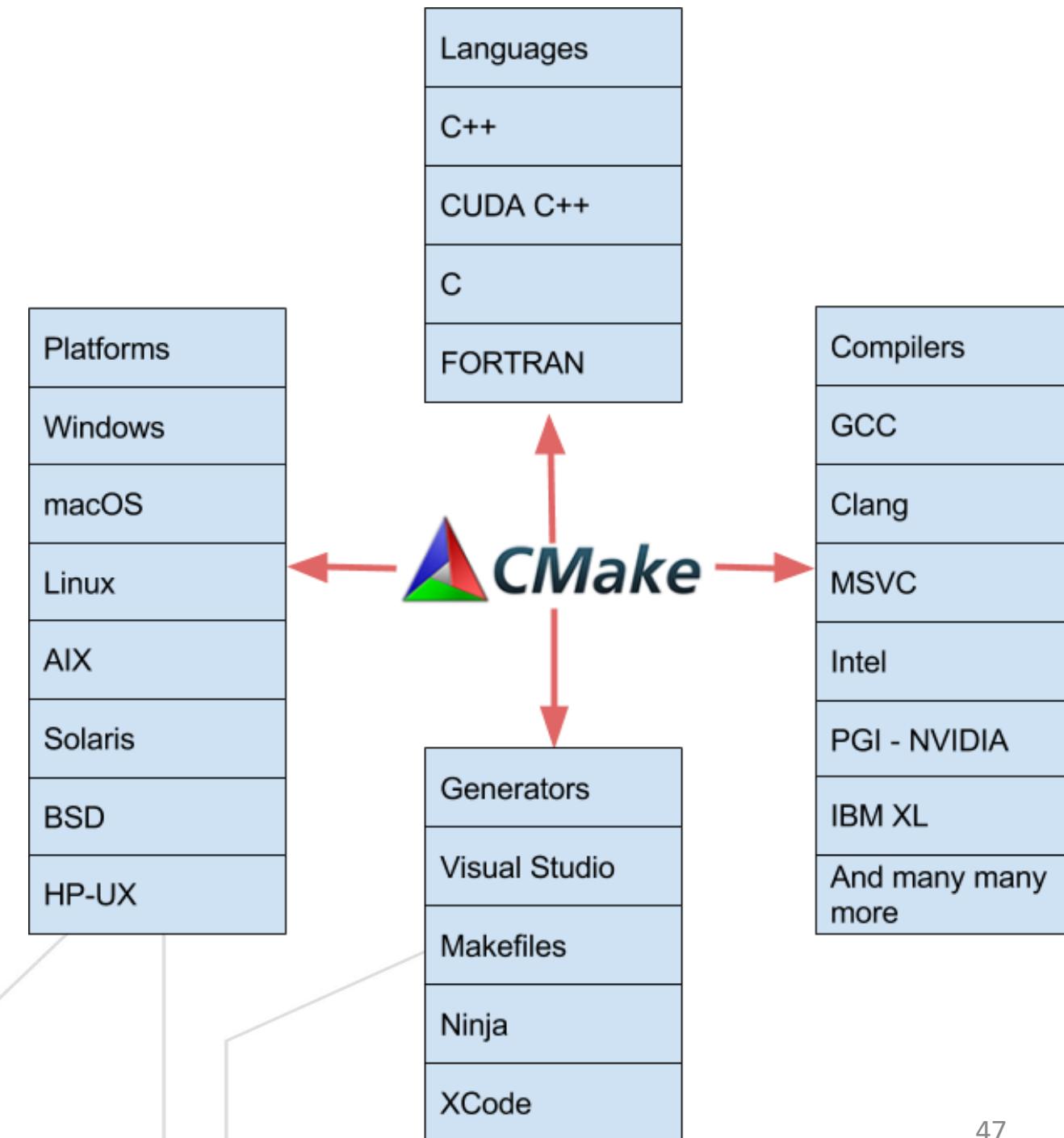
- nvcc直接编译即可
 - 与gcc/g++类似

● 使用Docker容器构建运行环境

- 留待TA罗柯同学介绍

● CMake

- 跨平台自动化建构系统



● CMake

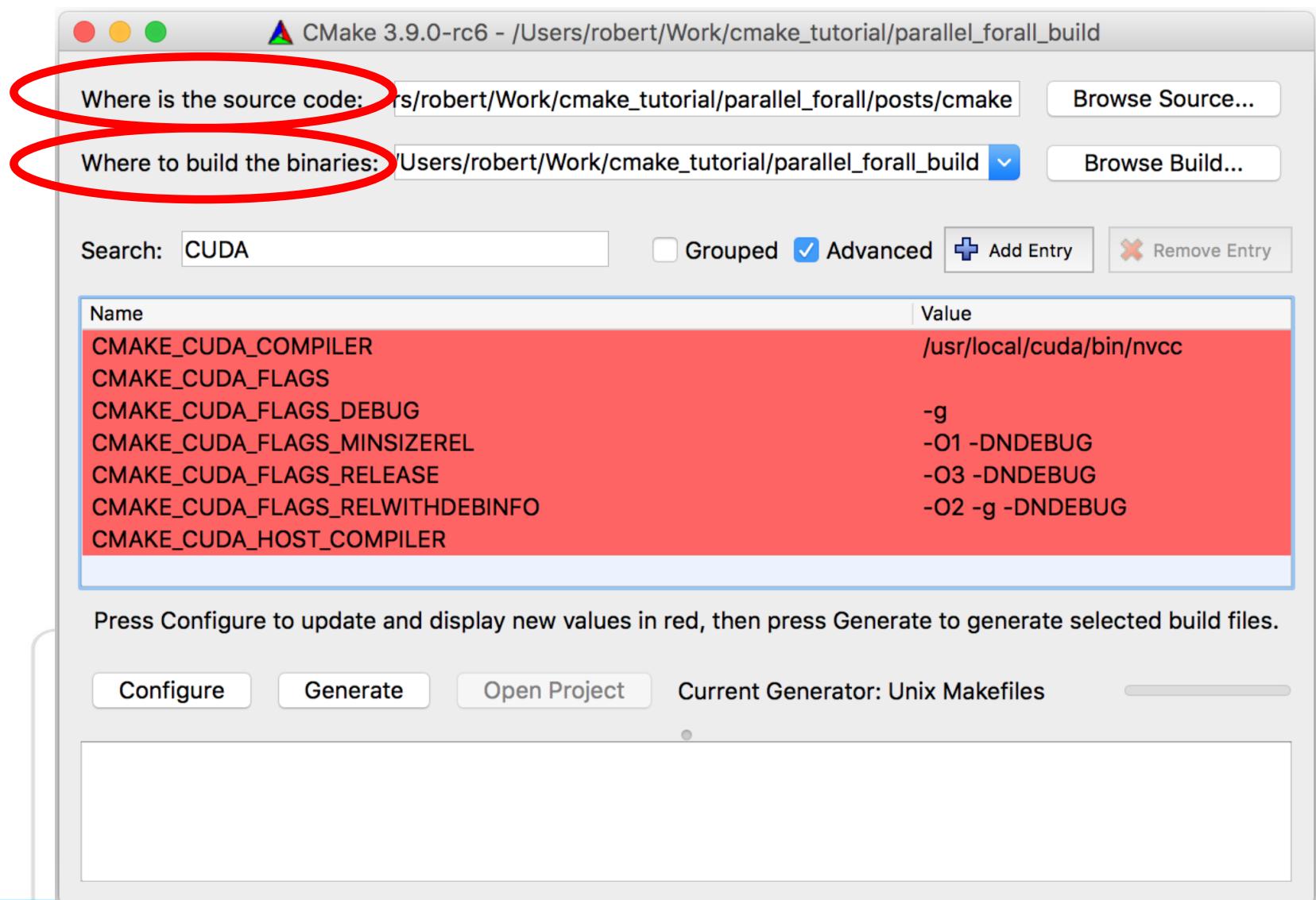
- 通过CMakeLists.txt 指明源文件、依赖关系、目标文件
 - 例子：

```
# 从helloworld.cu构建helloworld项目
# 检查cmake版本及自动寻找CUDA库路径
cmake_minimum_required(VERSION 2.8)
find_package(CUDA QUIET REQUIRED)

# 指明二进制可执行文件名称及源文件名称
cuda_add_executable(
    helloworld
    src/helloworld.cu)
```

● CMake

– 使用命令行或GUI创建项目文件



● 查询支持CUDA的硬件

- 源码

```
int nDevices;
cudaGetDeviceCount(&nDevices);

for (int i = 0; i < nDevices; i++) {
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, i);
    printf(" Device Number: %d\n", i);
    printf(" Device name: %s\n", prop.name);
    printf(" Memory Clock Rate (KHz): %d\n", prop.memoryClockRate);
    printf(" Memory Bus Width (bits): %d\n", prop.memoryBusWidth);
    printf(" Peak Memory Bandwidth (GB/s): %f\n\n",
           2.0*prop.memoryClockRate*(prop.memoryBusWidth/8)/1.0e6);
}
```

- 输出

```
Device Number: 0
Device name: Tesla C2050
Memory Clock Rate (KHz): 1500000
Memory Bus Width (bits): 384
Peak Memory Bandwidth (GB/s): 144.00
```

Questions?

