GPU加速FFT

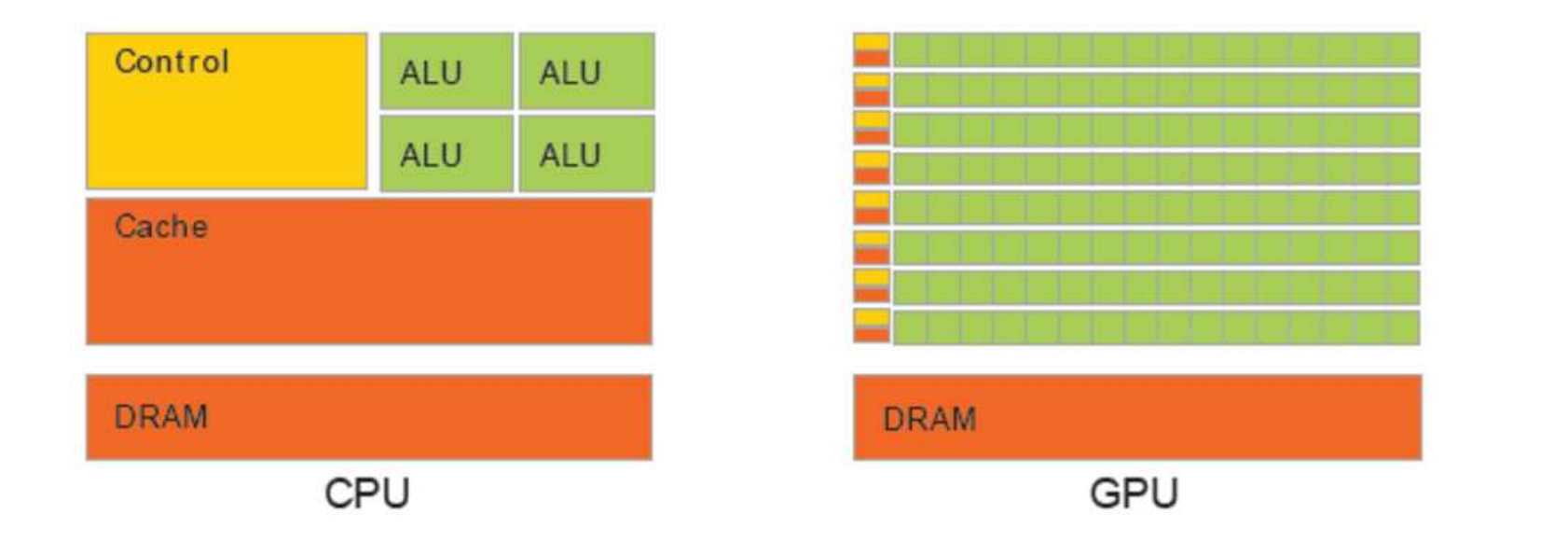
智能1602

金可欣

201608010605

GPU实现多线程并行运算

- CPU因为设计的原因而限制了核心数，所以为进程分配的资源也是有限的。
- GPU设计出来的初衷是用来进行显示信息的运算，拥有远大于CPU的核心数，可以实现高度并行运算。但最初也仅仅只能进行图形运算。



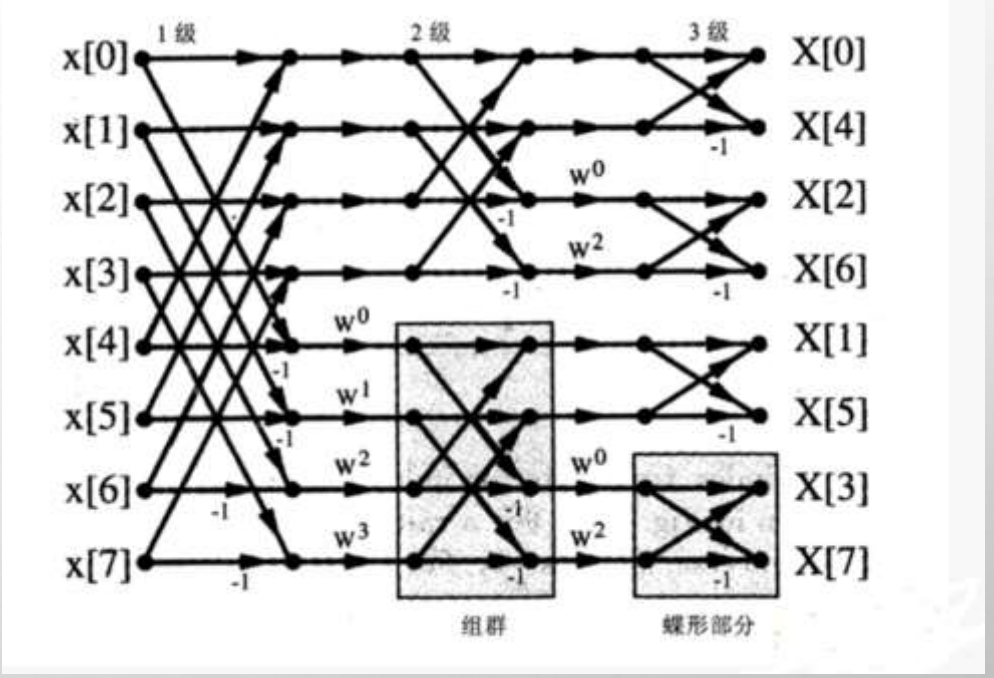
CUDA

- 随着CPU发展而算力并没有呈现预想中的速度的增长。于是，最初是由一群黑客，将运算任务伪装成图形计算发给了GPU，由GPU完成运算后再返回给内存。这大大提高了运算效率。
- 然后，英伟达公司便推出了CUDA，即将图形处理器GPU变为了一个可以处理高度并行运算的计算机硬件。

FFT

在CPU运算中，FFT通过递归实现每一轮的蝴蝶变换。

如果将其运用到GPU运算中，则每一阶的蝴蝶变换都可以交由不同线程运算（数据不冲突）而下一阶的则需要等待前一轮执行完才能分配线程。



核函数

核函数是GPU运算的核心，在CUDA应用中，核函数被打包成很多份，并发往各个就绪的线程，并进行运算

```
47  static __global__ void cufftComplexScale(cufftComplex *idata, cufftComplex *odata, const long int size, float scale)
48  {
49      const int threadID = blockIdx.x * blockDim.x + threadIdx.x;
50
51      if (threadID < size)
52      {
53          odata[threadID].x = idata[threadID].x * scale;
54          odata[threadID].y = idata[threadID].y * scale;
55      }
56  }
```

过程

申请设备存储空间

```
cufftComplex *data_dev; // 设备端数据头指针
cufftComplex *data_Host = (cufftComplex*)malloc(NX*BATCH * sizeof(cufftComplex)); // 主机端数据头指针
cufftComplex *resultFFT = (cufftComplex*)malloc(N*BATCH * sizeof(cufftComplex)); // 正变换的结果
cufftComplex *resultIFFT = (cufftComplex*)malloc(NX*BATCH * sizeof(cufftComplex)); // 先正变换后逆变换的结果

// 初始数据
```

运算

```
cudaMalloc((void**)&data_dev, sizeof(cufftComplex)*N*BATCH); // 开辟设备内存
cudaMemset(data_dev, 0, sizeof(cufftComplex)*N*BATCH); // 初始为0
cudaMemcpy(data_dev, data_Host, NX * sizeof(cufftComplex), cudaMemcpyHostToDevice); // 从主机内存拷贝到设备内存

cufftExecC2C(plan, data_dev, data_dev, CUFFT_FORWARD); // 执行 cuFFT, 正变换
cudaMemcpy(resultFFT, data_dev, N * sizeof(cufftComplex), cudaMemcpyDeviceToHost); // 从设备内存拷贝到主机内存

cufftExecC2C(plan, data_dev, data_dev, CUFFT_INVERSE); // 执行 cuFFT, 逆变换
cufftComplexScale << <dimGrid, dimBlock >> > (data_dev, data_dev, N, 1.0f / N); // 乘以系数
cudaMemcpy(resultIFFT, data_dev, NX * sizeof(cufftComplex), cudaMemcpyDeviceToHost); // 从设备内存拷贝到主机内存
```

总结

- 在CUDA中，英伟达公司制作了针对FFT的头文件，所以无需再去重新定义各种函数和运算。

只需要修改改变参数以及乘系数就可以了。为了比较，我准备了在CPU上跑的递归FFT算法并且比较其耗时。

数据规模	GPU时间(ms)	CPU时间(ms)
1<<12	6	4
1<<13	5	11
1<<14	6	19
1<<15	5	45
1<<16	5	94
1<<17	8	211