并行计算与GPGPU

石家庄一中 成羽丰

河北衡水中学 国家琦

并行计算与GPGPU的基本概念

概念简述

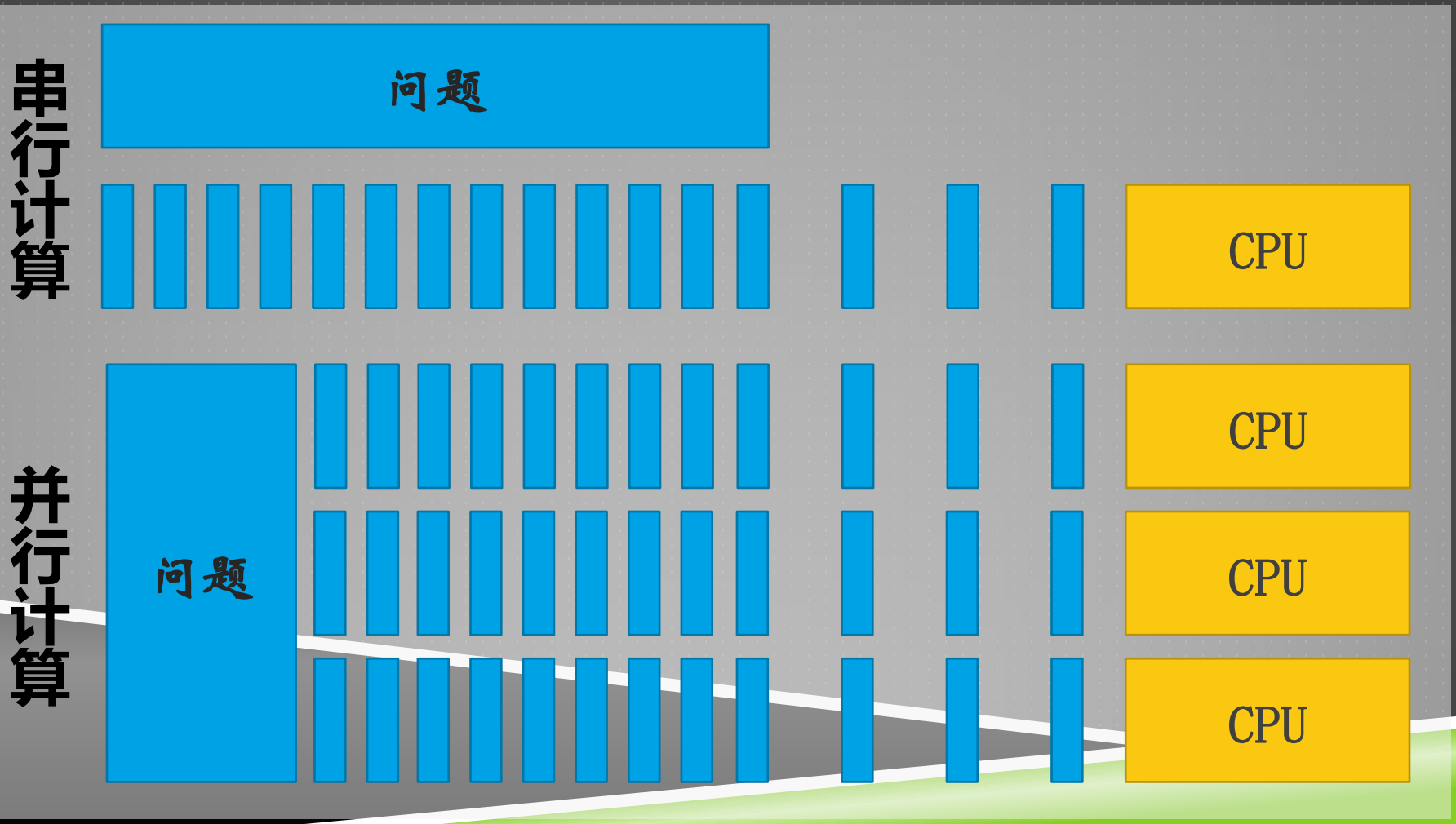
并行计算

- Long long ago……计算机出现了。
- 人们为计算机创造了各种算法，并将它们翻译为一条接一条地顺序执行的计算机指令。
- 经过简单分析可得出，此时计算机的运算速度就是执行每条指令所需平均时间*指令数。
- 执行每条指令所需平均时间与计算机的时钟频率成反比。
- 于是人们试图通过加快计算机频率的方式，优化执行每条指令所需平均时间。
- 加快计算机频率的方式起初很有成效，直到……

并行计算

- 计算机芯片的能耗$P = C \times V^2 \times F$，其中：
  - P代表计算机芯片的能耗
  - C代表每时钟周期变换的电容数量
  - V代表电压
  - F代表计算机的频率（每秒时钟周期数量）
- **增加频率势必会加大计算机的能耗!**
- 随着2004年Intel公司Tejas与Jayhawk计划的取消，传统的频率调解方法走上了死路。
- 于是人们转而开发**并行计算**在提高计算机性能方面的潜力。

并行计算



并行计算的类型

- 位级并行
- 指令级并行
- 数据级并行
  - 即将数据分散到不同处理器并行处理。
  - **循环并行化**：要求单次迭代不依赖于之前的迭代。
  - SIMD：对一组数据同时执行相同的操作。
- 任务级并行
  - **线程同步**：互斥锁、信号量、内存屏障、信号、管道。
  - **数据同步**：缓存一致性、数据复制。

GPGPU

- Long long ago，计算机的显示卡依赖于CPU进行图形运算。
- 随着多媒体技术的发展，为通用计算而设计的CPU愈发不能胜任繁杂的图形计算任务。
- 1999年，NVIDIA发布了GeForce256图形处理芯片标志了GPU的诞生。
- 显示卡利用专为图形计算设计的GPU，能够实现更有效的图形运算。
- GPU的一大特点是非常适合于并行计算。
- 人们为了充分利用GPU强大的并行处理能力，又提出了GPGPU的概念。

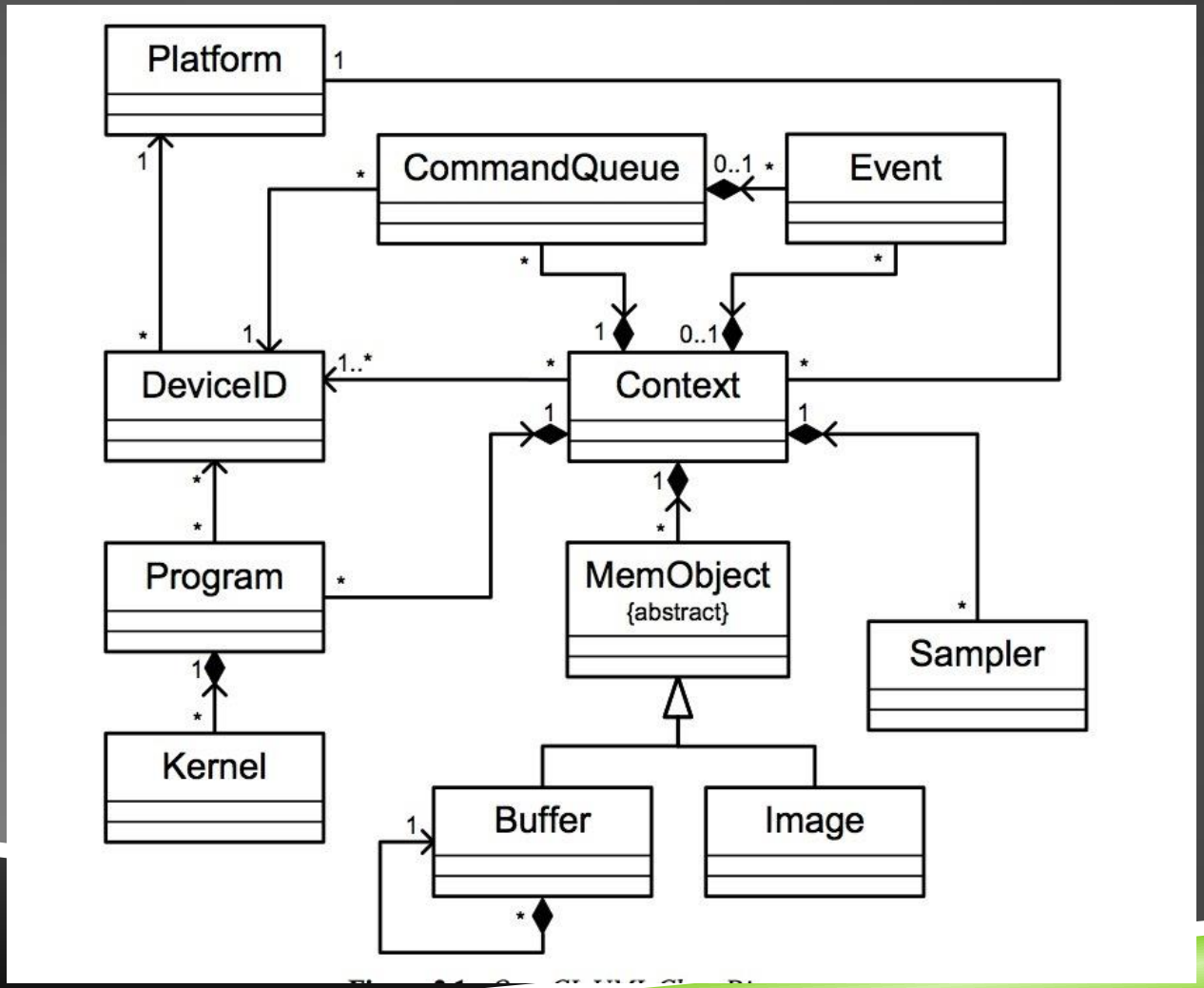
GPGPU

- GPGPU(通用图形处理，General-purpose computing on graphics processing units)是利用GPU代替CPU进行部分通用计算任务的技术。
- 它利用现代GPU强大的并行处理能力和可编程流水线，使得非图形处理任务可以并行地运行于GPU之上。
- 目前，GPGPU框架主要有NVIDIA推出的CUDA，AMD发展的AMD APP，微软研发的DirectCompute和起初由Apple开发OpenCL。
  - 这里我选用了较为开放与通用的OpenCL。

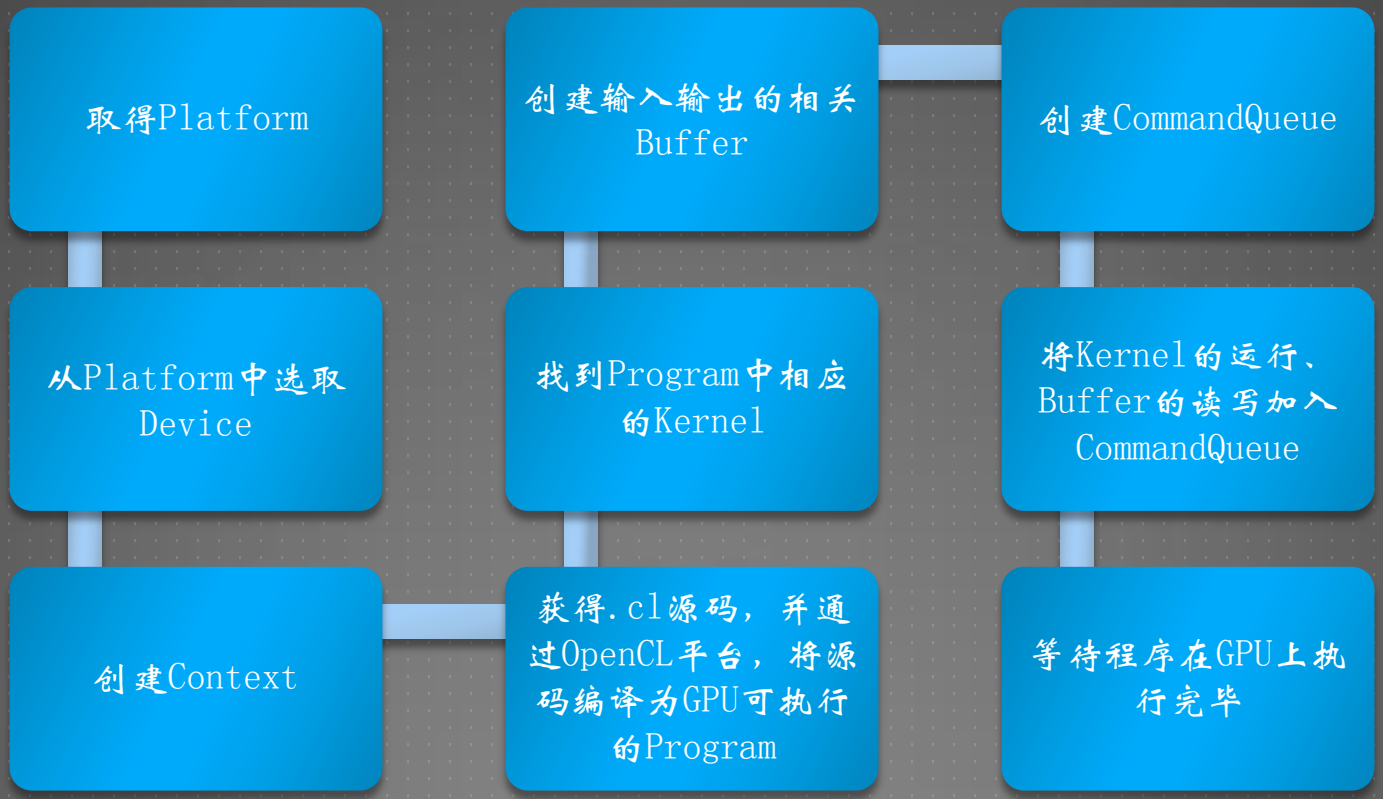
OpenCL

OpenCL中主要概念有

- Platform
- Device
- Program
- Kernel
- CommandQueue
- Context
- MemObject
- Event



OpenCL编程流程


取得Platform
从Platform中选取Device
创建Context
获得.cl源码，并通过OpenCL平台，将源码编译为GPU可执行的Program
找到Program中相应的Kernel
创建输入输出的相关Buffer
创建CommandQueue
将Kernel的运行、Buffer的读写加入CommandQueue
等待程序在GPU上执行完毕

一段简单的OpenCL程序

```
#include <CL/cl.h>
#include <cstring>
#include <cstdio>
char src[]={"Hello"};
char dst[6];
char clSource[]={
       "kernel void calc(global char* src,global char* dst){"
       "    int myID=get_global_id(0);"
       "    dst[4-myID]=src[myID];"
       "}"
};
size_t length=5;
const char * source=clSource;
size_t sourceL=strlen(clSource);
```

一段简单的OpenCL程序

```
int main(){
        cl_platform_id platform;
        clGetPlatformIDs(1,&platform,NULL);

        cl_device_id device;
        clGetDeviceIDs(platform,CL_DEVICE_TYPE_GPU,1,&device,NULL);

        cl_context context;
        Context=clCreateContext(NULL,1,&device,NULL,NULL,NULL);

        cl_program program;
        program=clCreateProgramWithSource(context,1,&source,&sourceL,NULL);
        clBuildProgram(program,1,&device,NULL,NULL,NULL);

        cl_kernel kernel=clCreateKernel(program,"calc",NULL);
```

一段简单的OpenCL程序

```
cl_mem inputBuffer;
inputBuffer=clCreateBuffer(context,CL_MEM_COPY_HOST_PTR,5,src,NULL);
cl_mem outputBuffer;
outputBuffer=clCreateBuffer(context,CL_MEM_WRITE_ONLY,5,NULL,NULL);

cl_command_queue queue=clCreateCommandQueue(context,device,0,NULL);

clSetKernelArg(kernel,0,sizeof(cl_mem),(void*)&inputBuffer);
clSetKernelArg(kernel,1,sizeof(cl_mem),(void*)&outputBuffer);

clEnqueueNDRangeKernel(queue,kernel,1,NULL,&length,NULL,0,NULL,NULL);
clEnqueueReadBuffer(queue,outputBuffer,CL_TRUE,0,5,&dst,0,NULL,NULL);

printf("原字符串：%s\n",src);
printf("翻转后：%s\n",dst);
```

一段简单的OpenCL程序

```
        clReleaseCommandQueue(queue);
        clReleaseMemObject(inputBuffer);
        clReleaseMemObject(outputBuffer);
        clReleaseKernel(kernel);
        clReleaseProgram(program);
        clReleaseContext(context);
        return 0;
}
```

输出：

原字符串：Hello
翻转后：olleH

一些利用GPGPU进行的并行计算试验

实战应用

试验计算机配置

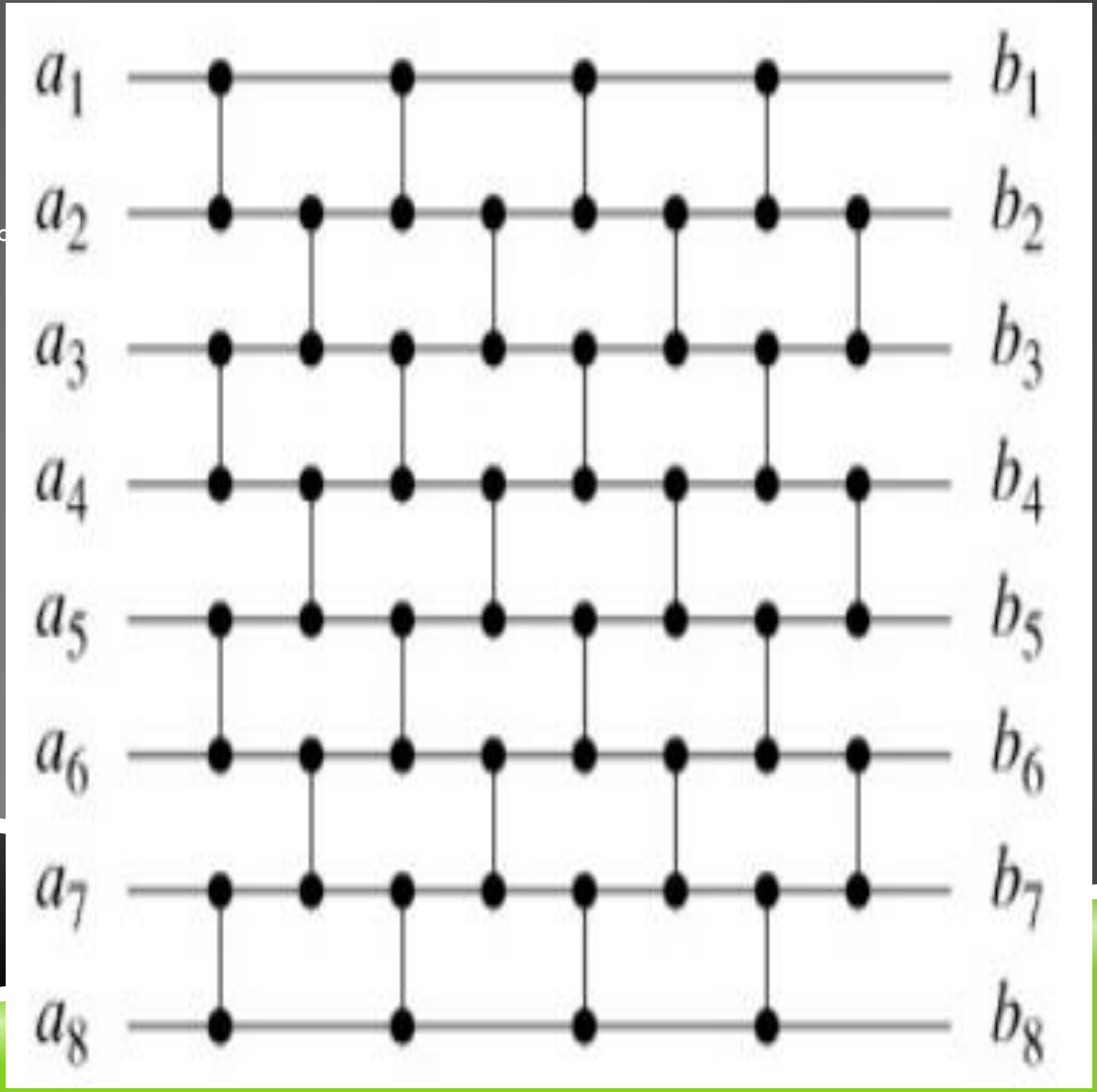
- 所有试验均运行于普通计算机之上，具体配置为：
  - CPU：AMD A6-3420M APU with Radeon™ HD Graphics 四核四线程
  - 内存：DDR3 1333MHz 4GB
  - 显卡：AMD Radeon HD 6400M
  - 显存：GDDR3 1GB
  - 操作系统：Microsoft Windows 8 RTM
  - 编译器及IDE：Visual Studio 2012 Professional

实例1 并行排序

- 排序问题是计算机经典算法之一。
- 基于比较的串行排序算法，最优理论复杂度只能做到$O(N\log N)$。
- 而通过算法的并行化，我们可以实现更低的时间复杂度。
- 并行排序主要有以下两种思路：
  - 一种是将已有的串行排序算法直接并行化，例如在二叉树上模拟快速排序算法，配合特殊的硬件，可以得到期望复杂度$O(\log N)$的良好算法。而将归并排序算法并行化，则可以轻易得到$O(N)$乃至更好的时间复杂度。
  - 另一种则是基于排序网络的。

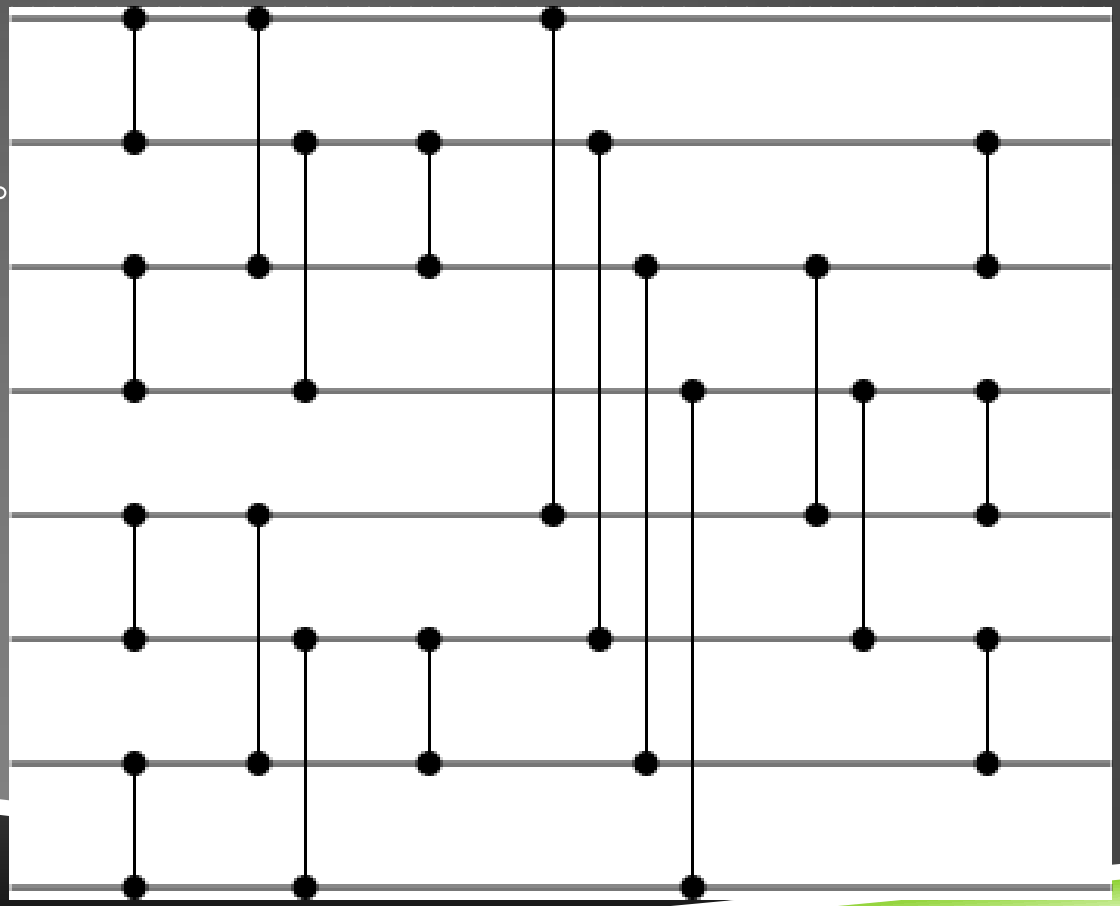
实例1 并行排序 – 排序网络

- 这是奇偶移项排序网络
- 它有$O(N^2)$个比较器。
- 拥有 $O(N)$ 的时间复杂度。



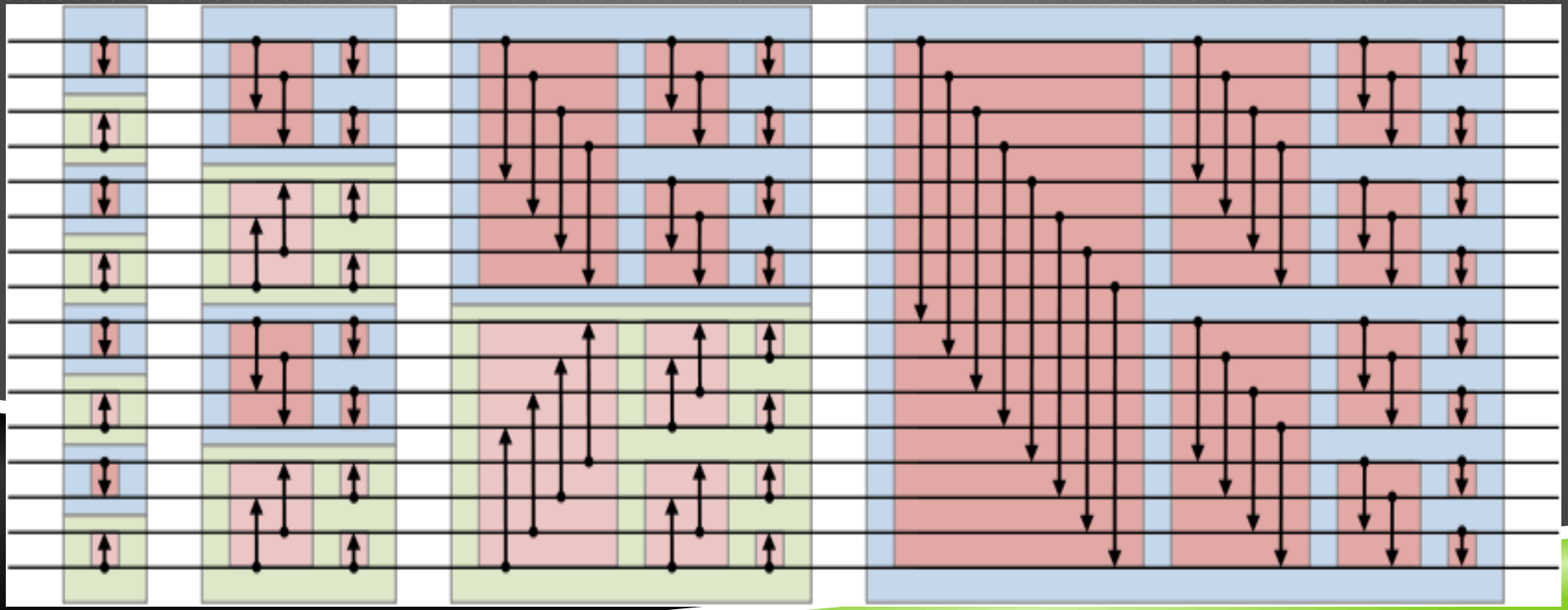
实例1 并行排序 – 排序网络

- 这是Batcher奇偶归并排序网络
- 它有 $O(N\log^2 N)$ 个比较器。
- 拥有$O(\log^2 N)$的时间复杂度。



实例1 并行排序 – 排序网络

- 这是双调排序网络
- 它有$O(N\log^2 N)$个比较器。
- 拥有$O(\log^2 N)$的时间复杂度。



实例1 并行排序 - 测试与结果

- 广告：并行排序试验数据由石家庄一中孙嘉裕同学赞助提供。
- 为了试验并行计算对排序的优化效果，我们先后做了如下试验：
  - 使用OpenCL编写朴素的并行归并排序算法，期望时间复杂度$O(N)$。
  - 排序2^{25}个数字：串行快速排序4.58s，并行归并排序2.22s。
  - 排序2^{26}个数字：串行快速排序9.71s，并行归并排序4.61s。
  - 而后又尝试了$O(\log^2 N)$的并行归并排序，但由于受到实际运算核心数等硬件限制，并没有取得预期的效果。
- GPGPU并不适合于排序等数据密集型任务，而是更加适用于计算密集型任务。

实例2 对MD5的暴力破解

- MD5是一种加密Hash算法，具有如下特点：
  - 对于任意字节序列，计算其Hash值很容易。
  - 求出一段字节序列，使得其Hash为给定值，很困难。
  - 对于字节序列的任何更改，都极大可能改变其Hash值。
  - 很难找到两段字节序列，拥有相同的Hash值。
- 由于MD5的上述特点，一些旧的用户数据库常常用它代替明文，存储用户密码的Hash。
- 暴力破解MD5即对于给定的一些Hash值，不断枚举明文，计算Hash，最终为尽可能多的Hash值找到可能的明文。
- 可以看出，暴力破解的过程拥有很大并行性。

实例2 对MD5的暴力破解 - 程序实现

- 程序大体思路如下：
  - 读入待破解的所有Hash。
  - 由短到长顺次枚举特定字符集的每个字串。
  - 计算字串的Hash，并在读入的列表中寻找此Hash，如果找到则输出。
- 具体细节：
  - 为了方便查找，先将输入的Hash排序并去重。
  - 查找时，由于输入已排序，所以采用了二分查找。
  - 为了减少CPU与GPU通信代价，字串的枚举、计算Hash与查找均运行于GPU一端。
  - 将每一字串按照特定的算法赋予唯一的编号，CPU负责协调GPU的不同线程应破解的字串区间。
- 设置了CPU对照组，以近乎相同的方法，暴力破解MD5.

实例2 对MD5的暴力破解 - 运行效率

	CPU	GPU
计算字串个数/s	1×10^6	2×10^7
最佳线程数量	4	512

可以看出，GPU的破解效率达到了CPU的20倍以上。

实例2 对MD5的暴力破解 - 结果分析

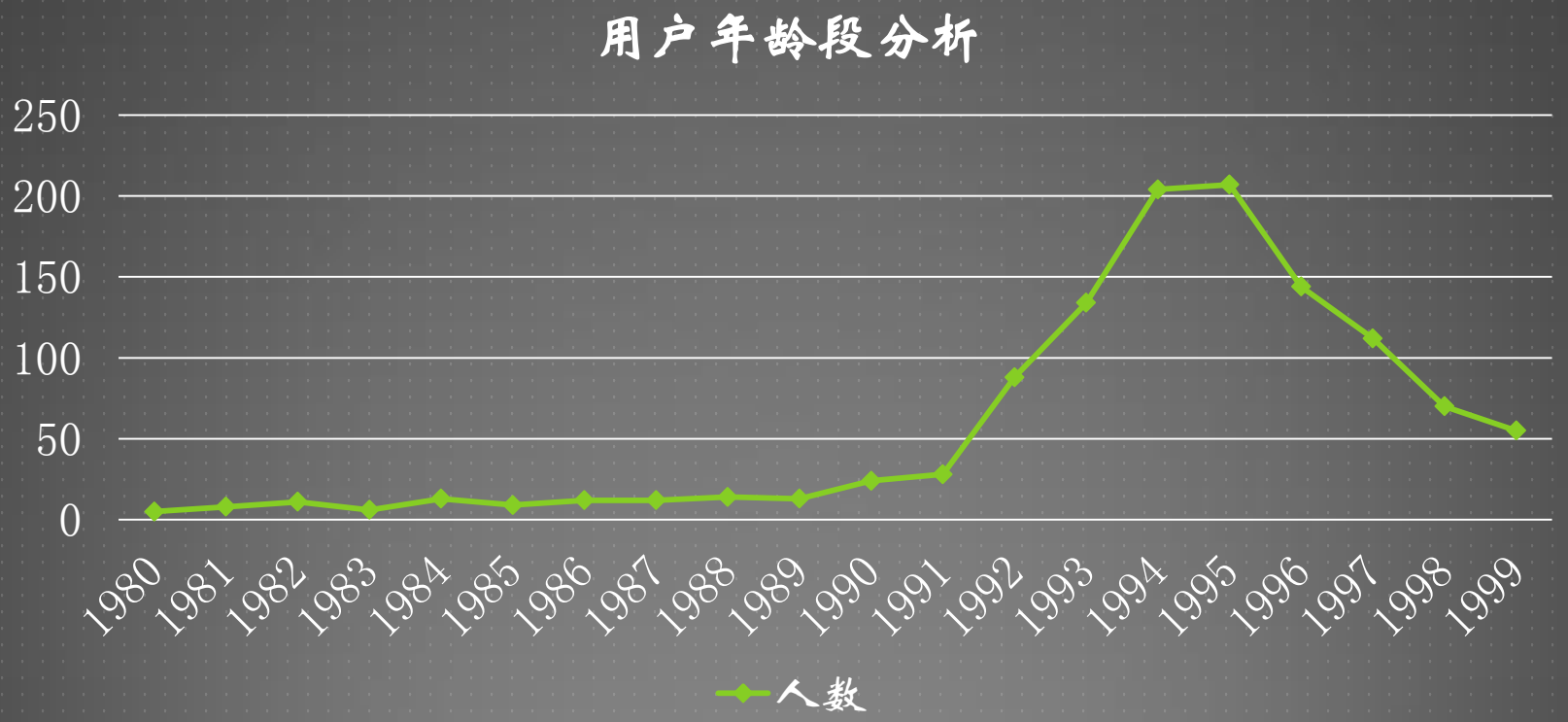
- 对于实验所用用户数据库，破解结果如下：
- 数据库中共29697名用户，19955个不重复的密码，平均每个密码有1.5名用户同时使用。
- 对于各种字符集分别进行了破解，累计用时大约在6h左右，最终破解出10054个密码，占密码总量的50.4%。
- 数据库中弱密码现象非常普遍。有1123人使用了纯生日密码。
- 另有424人使用了手机号码，其中
  - 中国移动用户338人
  - 中国联通用户62人
  - 中国电信用户24人

实例2 对MD5的暴力破解 – 结果分析

使用人数排名前10的密码

排名	明文	使用人数
1	123456	1197
2	1029384756	274
3	111111	154
4	123	151
5	123123	92
6	000000	87
7	1	86
8	123456789	67
9	12345	54
10	123321	49

实例2 对MD5的暴力破解 - 结果分析



实例2 对MD5的暴力破解 - 总结

- 虽然试验结果较为理想，但还存在很大的改进空间：
  - 输入Hash存于GPU的Global内存空间，访问较慢，二分查找时访问效率低。
  - GPU所用DRAM，适合于线性访问，随机访问开销较大，二分查找效率进一步下降。
  - GPU计算过程中，CPU处于闲置状态，很大地浪费了计算资源。
  - 已破解的Hash没有从数组中清除出去。
  - 逐个枚举的暴力方法，命中率不如结合模式或字典高。
  - 时空复杂度不平衡，可采用彩虹表等方式，平衡空间与时间，加速破解。

参考资料

- http://en.wikipedia.org/wiki/Parallel_computing
- http://en.wikipedia.org/wiki/GPGPU
- http://en.wikipedia.org/wiki/Batcher_odd-even_mergesort
- http://en.wikipedia.org/wiki/Bitonic_sorter
- http://www.idnovo.com.cn/hardware/2010/0906/article_1972.html
- http://www.matrix67.com/blog/archives/185
- http://baike.baidu.com.cn/view/5580839.htm

谢谢大家！