GPU指令执行过程（这里指令指的是类似于

Computing y = ax + y in parallel using CUDA: 
global 
void n, float 
alpha, float *x, 
int i 
blockIdx.x*b10ckDim.x 
threadldx.x; 
if( i<n ) y C i] 
alpha*xCi] 
float *Y) 
// Invoke parallel SAXPY kernel (256 threads per block) 
int rob locks 
(n 
255) / 256; 
saxpy_parallel<<<nblocks, 2.0, x, y); 

的指令模块，把每一个类似上面的一次循环的内容叫做操作）

指令到GPU时，我们找到一个warp（warp是同时执行同一条指令的线程的集合，而线程是执行一条指令的最小单元，在这里理解成执行一次操作）

然后考虑barrier synchronization

（这里多了一个概念，CTA（cooperative thread array），同时处理一个程序，最终一起得到一个结果）

（实验从这里开始，上面的调度不是我们能搞定的，但是写进论文吹水）

先说逻辑，实现先不管。

对于一个特定的warp，该warp含有多个SP（stream processor，拥有完整的处理整数和浮点数运算的能力），每个SP中有多个线程。（我感觉是这样的，这个warp中的所有线程同时进行运算）

分出SP的目的是便于接线和管理，你看下面这个图：

Work 
dtp ut 
Tatu re 

如果直接管理几百个个线程接线可能会接死人，所以分出了SP这一中间层次。

SFU先不管，书上写的是求倒数什么的，我们写成矩阵运算。

然后这个warp中的数据分4种：

1. global，外部数据，主要是要考虑读写顺序
2. shared，对于这个CTA里面的线程可见的数据，有多个独立单元，同时为了避免冲突，要一个方阵储存数据（interconnection network），详细一点说，就是因为shared memory分成了16个bank（理解成16堆不重复的数据），每一个bank只有一个数据输出线路，因为同时可能读取16个地址，会造成冲突（同时读取同一个bank），为了完全避免冲突，将每一个bank复制16次，构造一个16X16的网络。
3. local，在外部的数据，只有对应线程能访问
4. constant，存放常量

global和shared的区别：

global对于所有CTA可见，这样就带来一个直接的问题，多个线程对于数据的操作是否具有一致性。生成期为机器运行时期。延时高

shared只对于当前CTA可见，延时低，生存期为程序运行时期。

这次实验只实现shared和constant存储。

每一个RF会提供1024个32位的寄存器

然后就到了SP内部运算，获取数据后运算，输出，单个运算之间不存在并行，包括浮点

指令集设计：

初步，不用SFU的时候的基本指令

12种，22条指令（每种指令带符号整数和浮点各一条，为了方便，统一使用第开头第一位位来区分，带符号整数为1，浮点为0）

指令格式为

Opcode.type d, a, b, c

操作数第一位指示是否为常数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation Code(10) | D(11) | A(11) | B(11) | C(11) |

或

|  |  |  |  |
| --- | --- | --- | --- |
| Operation code(10) | D（11） | A（11） | immediate（32） |

D,a,b,c都是10位寄存器地址

1. Add.type
   1. Add.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000000 | d | A | B | - |

1. Add.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000000 | D | A | B | - |

1. Sub.type
   1. Sub.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000001 | D | A | B | - |

1. Sub.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000001 | D | A | B | - |

1. Mul.type
   1. Mul.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000010 | D | A | B | - |

1. Mul.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000010 | D | A | B | - |

1. Abs.type
   1. Abs.s32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000011 | D | A |  | - |

1. Abs.f32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000011 | D | A |  | - |

1. Neg.type
   1. neg.s32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000100 | D | A |  | - |

1. neg.f32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000100 | D | A |  | - |

1. Min.type
   1. Min.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000101 | D | A | B | - |

1. Min.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000101 | D | A | B | - |

1. Max.type
   1. Max.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000110 | D | A | B | - |

1. Max.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000110 | D | A | B | - |

1. not.type
   1. Min.s32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000000111 | D | A | - | - |

1. Min.f32 d, a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000000111 | D | A | - | - |

1. and.type
   1. and.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000001000 | D | A | B | - |

1. and.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000001000 | D | A | B | - |

1. or.type
   1. or.s32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000001001 | D | A | B | - |

1. or.f32 d, a, b

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000001001 | D | A | B | - |

1. Ld.shared（不区分，都是32位）
   1. Ld.shared d, a, imm

|  |  |  |  |
| --- | --- | --- | --- |
| 0000001010 | D | A | Imm |

D = \*(A+imm)

1. St.shared（不区分，都是32位）
   1. St.shared d, a, imm

|  |  |  |  |
| --- | --- | --- | --- |
| 0000001011 | D | A | imm |

\*(d+imm) = a

1. Bar.sync d

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000001100 | D |  |  | - |

四周期：

1. 取指令，选择空闲SP
2. 译码，读数据，将常数放到constant memory里面
3. 运算
4. 输出结果