论程序底层优化的一些方法与技巧

成都七中　骆可强

？

=时间效率+ 空间效率

T(n)=算法(f(n))

$(\exists n_0, c \in N^+)$

$T(n) \le C1 \times C2 \times f(n)$ $(\forall n > n_0)$

底层优化　　算法

一个简单的例子: 求最大值

```
// 第六次优化
int get_max(int* a,int l){
        assert(l%4==0);
        assert(sse4);
        int ret,tmp[4];
        __asm__ __volatile__ (
                        "\txorps        %%xmm0,         %%xmm0\n"
                        "LP4:\n"
                        "\tpmaxsd       (%1),         %%xmm0\n"
                        "\taddl         $16,          %1\n"
                        "\tsubl         $4,           %2\n"
                        "\tjnz          LP4\n"
                        "\tmovdqu       %%xmm0,       (%3)\n"
                        "\tmovl         (%3),         %%eax\n"
                        "\tcmpl         4(%3),        %%eax\n"
                        "\tcmovll       4(%3),        %%eax\n"
                        "\tcmpl         8(%3),        %%eax\n"
                        "\tcmovll       8(%3),        %%eax\n"
                        "\tcmpl         12(%3),       %%eax\n"
                        "\tcmovll       12(%3),       %%eax\n"
                        "\tmovl         %%eax,        %0\n"
                        :"=m"(ret)
                        :"r"(a),"r"(l),"r"(tmp)
                        :"%eax");
        return ret;
}
```

编号	平均 时钟周期	优化 (%)	优化方法
0	7.53	—	—
1	6.60	12%	优化寻址
2	3.48	54%	多路求值
3	2.13	72%	内嵌汇编
4	1.74	77%	内嵌汇编 + 多路求值
5	1.59	80%	SIMD
6	?	?	SSE4

论文中所覆盖的主题

- CPU 指令运行的效率表现
  - 浮点
- 数值运算的优化
  - 除法
    - 除常数
    - ……
  - 乘法
  - 高精度
- CPU 优化特性
  - 缓存机制
  - 分支预测
  - 乱序执行
- 位运算技巧
  - 位压缩
  - 打包统计
  - 消除分支
- 高维数组的使用
  - 寻址
  - 底层表现

高维数组访问的底层表现

```
int a[2049][2049];
int main(){
        int i,x,y;
        for(i=0;i<100;i++){
                for(y=0;y<2049;y++)
                        for(x=0;x<2049;x++)
                                a[x][y]=x+y;
        }
        return 0;
}
```

- 尽量按照数据在内存中放置顺序去访问它
- 避免高维数组的尺寸是 2 的方幂

Time1 ： 1.760s
Time2 ： 18.757s
Time3 ： 4.644s

$A_{1,1}$	$A_{1,2}$	……	$A_{1,2048}$	$A_{2,1}$	……

编译器对除以常数的优化

a /= 7

$$(a \times (613566757 + 2^{32})) >> 35$$

- 除以常数比除以变量更快
- 在某些场合，我们需要多次除以同一个变量，也可以事先计算出这些常数起到优化效果

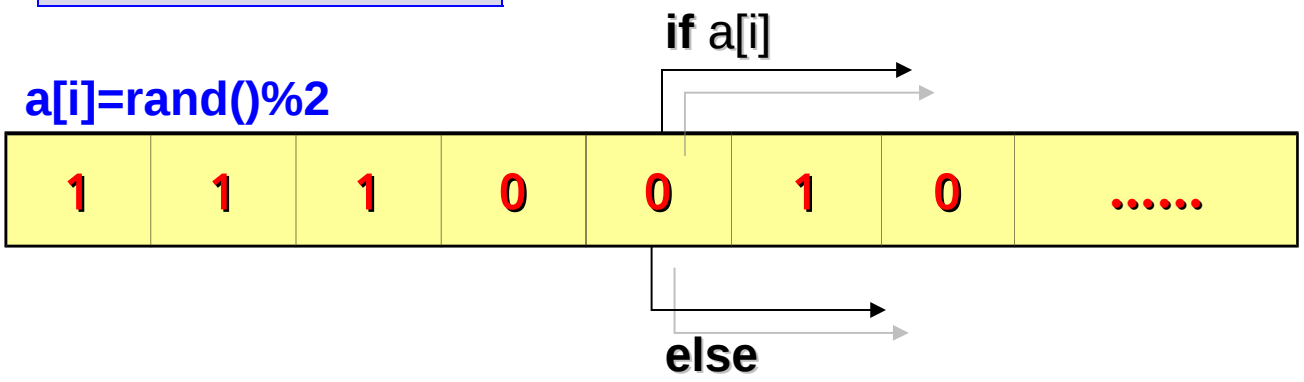
```
movl    $613566757, %edx
movl    %ecx, %eax
mull    %edx
subl    %edx, %ecx
shrl    %ecx
addl    %ecx, %edx
shrl    $2, %edx
movl    %edx, -8(%ebp)
```

CPU 的分支预测机制

```
int t=0;
for(int i=0;i<N;i++){
        if(a[i])t+=1;
        else t+=2;
}
```

测量结果 1 ： 3.2×10^7 个时钟周期

测量结果 2 ： 1.02×10^8 个时钟周期



消除条件分支

```
int cmp(int a){return (a>>31)+(-a>>31&1);}
```

```
int abs(int x){int y=x>>31;return (x+y)^y;}
```

```
x^=a^b
```

在信息学奥赛中的实践

题目:	麦森数 (mason)
来源:	NOIP2003
算法:	朴素的高精度计算
测试情况:	70 分
优化:	消除除法与条件分支
优化情况:	100 分

题目:	瑰丽的华尔兹 (adv1900)
来源:	NOI2005
算法:	朴素的动态规划， $O(N^4)$
测试情况:	60 分
优化:	优化高维数组寻址
优化情况:	100 分（均在 0.5s 内出解）

题目:	翻译玛雅著作 (writing)
来源:	IOI2006
算法:	O(字符集 × 串长) 的枚举
测试情况:	70 分
优化:	汇编优化循环与数组访问
优化情况:	100 分

总结

过早的优化是效率低下的根源

程序的优化是无止境的

Keep It Simple and Stupid

谢谢大家

欢迎提问