CSAPP讨论（七）

优化程序性能

马玉坤

2016年11月20日

程序性能优化的途径

- 算法与数据结构，优化时间复杂度
- **采用正确的写法，优化常数（本章的主要内容）**
  - 针对编译器
  - 针对CPU
- 并行计算

常数优化的危险

```
void twiddle1(int *xp, int *yp) {
  *xp += *yp;
  *xp += *yp;
}
void twiddle2(int *xp, int *yp) {
  *xp += 2 * *yp;
}
```

twiddle2与twiddle1应该等价?

常数优化的危险

```
void twiddle1(int *xp, int *yp) {
  *xp += *yp;
  *xp += *yp;
}
void twiddle2(int *xp, int *yp) {
  *xp += 2 * *yp;
}
```

twiddle2与twiddle1应该等价?

- 然而并不。

常数优化的危险

```
void twiddle1(int *xp, int *yp) {
  *xp += *yp;
  *xp += *yp;
}
void twiddle2(int *xp, int *yp) {
  *xp += 2 * *yp;
}
```

twiddle2与twiddle1应该等价?

- 然而并不。
- 考虑*xp与*yp相等的情况（即xp与yp指向同一个地址）。

常数优化的危险

```
void twiddle1(int *xp, int *yp) {
  *xp += *yp;
  *xp += *yp;
}
void twiddle2(int *xp, int *yp) {
  *xp += 2 * *yp;
}
```

twiddle2与twiddle1应该等价?

- 然而并不。
- 考虑*xp与*yp相等的情况（即xp与yp指向同一个地址）。
- 盲目优化会带来潜在的风险（无论对编译器还是程序员）。

常数优化的威力

Result	Memory	Time	Language	Code_Length
Time_Limit_Exceed	13020 kb	6184 ms	C++/Edit	3344 B
Accepted	13020 kb	736 ms	C++/Edit	3372 B

- 上面的代码没有加inline
- 下面的代码加了inline

程序性能表示

CPE

- Cycles Per Element

程序性能表示

CPE

- Cycles Per Element
- 与大O表示法的比较

程序优化的途径(1)

代码移动

- 将频繁计算但值不会变化的量提前算好，存到变量中
- `for (int i = 0; i < strlen(s); i++)`

使用临时变量存储中间结果

- 需要将某个元素*x频繁修改，但只在意其结果
- 使用中间变量
- 寄存器=*x
- 对寄存器多次修改
- *x=寄存器

现代处理器

乱序

- 指令并行
- 遇到分支怎么办?

现代处理器

乱序

- 指令并行
- 遇到分支怎么办?
- 分支预测
- 预测成功：指令进入退役单元，程序状态（寄存器等）更新
- 预测失败：从预测处重新执行

现代处理器

功能单元性能

Table: Intel Core i7算术运算性能

运算	整数		单精度		双精度	
	延迟	发射	延迟	发射	延迟	发射
加法	1	0.33	3	1	3	1
乘法	3	1	4	1	5	1
除法	11-21	5-13	10-15	6-11	10-23	6-19

循环结构效率的瓶颈

寄存器

- 只读寄存器:只用作源值
- 只写寄存器:只用作数据传送
- 局部寄存器:迭代与迭代之间不相关
- 循环寄存器:一次迭代中产生的值会在另一次用到

循环结构效率的瓶颈

寄存器

- 只读寄存器:只用作源值
- 只写寄存器:只用作数据传送
- 局部寄存器:迭代与迭代之间不相关
- 循环寄存器:一次迭代中产生的值会在另一次用到

瓶颈寄存器

循环寄存器的操作链

程序优化的途径(2)

循环展开

```
int i;
for (i = 0; i < len; i += 3) {
  res += a[i] + a[i+1] + a[i+2];
}
for (; i < len; i++) {
  res += a[i];
}
```

- 减少循环索引计算
- 减少条件分支
- 增加并行度
- CPE:1.0

程序优化的途径(2)

提高并行性

```
int i;
for (i = 0; i < len; i += 2) {
  res0 += a[i];
  res1 += a[i+1];
}
for (; i < len; i++) {
  res0 += a[i];
}
```

程序优化的途径(2)

重新结合变换

- 代码1`res = res + (data[i] + data[i+1])`
- 代码2: `res = (res + data[i])+ data[i+1]`
- SIMD:Single-Instruction, Multiple-Data
- 利用SIMD模型提高并行性
- 结合律

程序剖析

GPROF

- *gcc -O1 -pg prog.c -o prog*
- *./prog file.txt*

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
97.58	173.05	173.05	1	173.05	173.05	sort_words
2.36	177.24	4.19	965027	0.00	0.00	find_ele_rec
0.12	177.46	0.22	12511031	0.00	0.00	Strlen

其他优化技巧

利用高速缓存

- 原理：一次将一行内存里的内容加载到缓存中
- `struct P{int x,y}[1000];`
  instead of
  `int x[1000],y[1000];`
- 内存对齐：
  `int array[1024] __attribute__((aligned(64)));`
- 相邻放置相关代码

Q&A&Q