体系结构Lab4

罗兴攀 PB19051150

1. 循环体展开正确性

有两种方法证明循环体展开的正确性：

- 分析代码。例如 `daxpy()和daxpy_unroll()` 函数，分析代码可知，后者将循环体展开并没有修改每一个 `Y[i]` 的计算方式，其它两个函数同理

Plain Text | 复制代码

```
void daxpy(double *X, double *Y, double alpha, const int N)
{
    for (int i = 0; i < N; i++)
    {
        Y[i] = alpha * X[i] + Y[i];
    }
}
void daxpy_unroll(double *X, double *Y, double alpha, const int N)
{
    //2层展开
    for (int i = 0; i < N - 1; i += 2)
    {
        Y[i] = alpha * X[i] + Y[i];
        Y[i + 1] = alpha * X[i + 1] + Y[i + 1];
    }

    if (N % 2)
    {
        Y[N - 1] = alpha * X[N - 1] + Y[N - 1];
    }
}
```

- 代码用例证明。编写测试代码，取不同的N，分别使用普通函数和循环展开函数对相同的Y进行计算后比较是否相同。

Plain Text | 复制代码

```
bool same(double *Y1,double*Y2,const int N)
{
    bool res=true;
    for(int i=0;i<N;i++){
        if(Y1[i]!=Y2[i]){
            printf("%d,%f,%f\n",i,Y1[i],Y2[i]);
            res=false;
        }
    }
    return res;
}

void test(int N){
    printf("N=%d:\n",N);
    double *X = new double[N], *Y = new double[N], alpha = 0.5, beta = 0.
1;
    double *Y1=new double[N],*Y2=new double[N];
    //std::random_device rd;
    std::mt19937 gen(0);
    std::uniform_real_distribution<> dis(1, 2);
    for (int i = 0; i < N; ++i)
    {
        X[i] = dis(gen);
        Y[i] = dis(gen);
    }
    for(int i=0;i<N;i++){
        Y1[i]=Y[i];
        Y2[i]=Y[i];
    }
    daxpy(X, Y1, alpha, N);
    daxpy_unroll(X, Y2, alpha, N);
    if(same(Y1,Y2,N)){
        printf("daxpy true\n");
    }
    for(int i=0;i<N;i++){
        Y1[i]=Y[i];
        Y2[i]=Y[i];
    }
    daxsbxpxy(X, Y1, alpha, beta, N);
    daxsbxpxy_unroll(X, Y2, alpha, beta, N);
    if(same(Y1,Y2,N)){
        printf("daxsbxpxy true\n");
    }
    for(int i=0;i<N;i++){
        Y1[i]=Y[i];
```

```
45          Y2[i]=Y[i];
46      }
47      stencil(Y1, alpha, N);
48      stencil_unroll(Y2,alpha,N);
49      if(same(Y1,Y2,N)){
50          printf("stencil true\n");
51      }
52  }
53  int main()
54  {
55      int N = 9;
56      for(int i=9;i<40;i+=9){
57          test(i);
58          test(i+1);
59      }
60      return 0;
61  }
```

运行结果如下：

```
lumos@gem:~/lab4$ g++ test.cpp -o test;./test
N=9:
daxpy true
daxsbxpxy true
stencil true
N=10:
daxpy true
daxsbxpxy true
stencil true
N=18:
daxpy true
daxsbxpxy true
stencil true
N=19:
daxpy true
daxsbxpxy true
stencil true
N=27:
daxpy true
daxsbxpxy true
stencil true
N=28:
daxpy true
daxsbxpxy true
stencil true
N=36:
daxpy true
daxsbxpxy true
stencil true
N=37:
daxpy true
daxsbxpxy true
stencil true
```

结果显示，多组测试结果都显示展开前后计算结果相同。

2.统计结果

将三个统计结果分别把保存为 `stats1.txt,stats2.txt,stats3.txt` 执行 `run.sh` 获取相关结果。只保留第2行到第7行的结果。

run.sh　Plain Text　复制代码

```
for num in 1 2 3;do
    cat stats${num}.txt | grep system.cpu_cluster.cpus.cpi|awk '{if(NR>=2&&NR<=7){print $2}}'
    cat stats${num}.txt | grep simSeconds|awk '{if(NR>=2&&NR<=7){print $2}}'
    cat stats${num}.txt | grep system.cpu_cluster.cpus.numInsts|awk '{if(NR>=2&&NR<=7){print $2}}'
done
```

		CPI	执行时长	指令条数
O1	daxpy	1.781688	0.000036	80014
		2.235975	0.000034	60019
	daxsbxpxy	2.094203	0.000063	120017
		2.113784	0.000063	120017
	stencil	1.962549	0.000049	99997
		2.059339	0.000044	85003
增加SIMD	daxpy	1.781688	0.000036	80014
		1.82532	0.000037	80015
	daxsbxpxy	2.01089	0.00006	120017
		2.040936	0.000061	120017
	stencil	1.962549	0.000049	99997
		2.000529	0.000043	85003
O3	daxpy	1.822972	0.000018	40022
		1.920801	0.000019	40026

	daxsbxpxy	2.073769	0.000029	55023
		2.06373	0.000028	55029
	stencil	2.239504	0.000034	59999
		2.798872	0.000035	50013

结果分析总述（详细分析见后）

- O1优化下，第一个和第三个、个函数的性能因展开而提高（即使CPI升高，但总时长是更优的），第二个函数性能无明显变化（总时长无变化）。
- 增加SIMD，不管有没有展开，程序性能基本都有提升（CPI降低，指令条数不变，总执行时长降低）
- O3优化可以大幅减少指令条数，但CPI反而升高了，不过，由于指令条数的减少，O3优化的总执行时长依旧小于O1优化。

3.展开与性能

O1优化下，第一个和第三个函数的性能因展开而提高（即使CPI升高，但总时长是更优的），第二个函数性能无明显变化（总时长无变化）。

第二个函数总执行时长几乎无变化，考虑到的可能原因有：

1. 循环展开后，增加了语句的数目，增大了指令缓存的容量压力，上表看出，这一原因在该例中占比较小；
2. 这个函数中只有一个操作，在CPU流水线上的指令重叠将较小，因此展开后的指令重叠效果并不显著；

第三个函数由于存在从上一步计算结果取值的操作，在流水线中可以提前获得该数据，故增加循环展开优化后性能提升明显，且循环展开后，该函数的指令条数不升反降，故总执行时长更少。循环展开主要通过减少数据竞争来提高程序性能，而第三个原函数存在较多的数据竞争，因此优化效果明显。

4.simd与性能

由上述表格分析可知，增加SIMD，不管是对原函数还是展开函数，都能提高性能。原因是更多的硬件支持可以通过一次执行多个相同操作的指令来实现并行处理多个数据，从而进一步加快程序的执行速度。更多的SIMD可以减少了数据竞争(hazard）和控制竞争。

5.编译器优化与性能

从表中不难看出编译器O3优化尽管CPI似乎比O1更大，但O3优化可以大幅减少指令条数，进而可以时总执行时长减少。不难猜测，O3优化可能通过使用一些较复杂的指令替换了多个简单的指令，这使得尽管单条指令执行时间长，但程序总时长却是更优的。

6.手动循环展开的意义

理论上将手动循环展开是有意义的，从实际实验结果来看，也是有意义的，即使不带来更好的结果，也不会比原来的结果差。手动循环展开在部分情况下可能得到更好的结果，但也有可能和不展开几乎没有区别，即从总执行时间来看，手动循环展开即使不提高性能，也几乎不会降低性能，因此手动循环展开总体来看是有意义的。

和编译器优化相比，手动优化依然是有意义的，因为在同样的O3优化下，手动展开的结果可能比原始的好，至少没有比原来的差。因此即使有了编译器优化，适当使用手动优化依然是有意义的。

7.总结

- 手动优化有意义，在部分情况下性能提高，即使不提高也不会比不优化差
- O3优化大幅减少指令条数
- 增加SIMD可以降低CPI
- 最佳配置为：手动展开+O3优化+增加SIMD。可以尽可能获得较好的性能