高性能编程实验 #1

姜洋帆 17341068

2019 年 9 月 1 日

1 实验内容

将文件“性能优化实验分析.pdf”中讨论的程序编译运行，并记录运行时间，然后利用 Linux 性能剖析工具 perf 分析程序被加速的原因。

2 相关代码

实验 PDF 上给出一个图像旋转的代码 (相当于矩阵转置)，并逐步进行三次优化

2.1 原始代码

直接使用两重循环，对矩阵进行转置操作，当 dim 值较大时，cache 无法装下矩阵的整行元素，导致局部性不好

```
#include<stdio.h>
#define MAX 4096
#define RIDX(i, j, dim) ((i)*(dim) + (j))

typedef struct{
    int red;
    int green;
    int blue;
}pixel;

void naive_rotate(int dim, pixel *src, pixel *dst)
{
    int i, j;
    for (i = 0; i < dim; i++)
    for (j = 0; j < dim; j++)
        dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}

pixel src[MAX*MAX];
```

```
pixel dst[MAX*MAX];

int main()
{
    int i;
    for (i = 0; i < 20; i++)
        naive_rotate(MAX, src, dst);
    return 0;
}
```

2.2 第一次尝试

对矩阵数据进行分块，分为 4*4 的小块，代码如下

```
#include<stdio.h>
#define MAX 4096
#define RIDX(i, j, dim) ((i)*(dim) + (j))

typedef struct{
    int red;
    int green;
    int blue;
}pixel;

void rotate(int dim, pixel *src, pixel *dst)
{
    int i, j, ii, jj;
    for (ii = 0; ii < dim; ii+=4)
    for (jj = 0; jj < dim; jj+=4)
        for (i=ii; i < ii+4; i++)
        for (j=jj; j < jj+4; j++)
            dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}

pixel src[MAX*MAX];
pixel dst[MAX*MAX];

int main()
{
    int i;
    for (i = 0; i < 20; i++)
        rotate(MAX, src, dst);
    return 0;
```

```
}
```

2.3 第二次尝试

改变分块大小，采用 32*32 分块，并 4*4 路循环展开

```
#include<stdio.h>
#define MAX 4096
#define RIDX(i, j, dim) ((i)*(dim) + (j))

typedef struct{
    int red;
    int green;
    int blue;
}pixel;

void rotate(int dim, pixel *src, pixel *dst)
{
    int i, j, ii, jj;
    for (ii = 0; ii < dim; ii+=32)
    for (jj = 0; jj < dim; jj+=32)
        for (i=ii; i < ii+32; i+=4)
        for (j=jj; j < jj+32; j+=4) {
            dst[RIDX(dim-1-j, i, dim)]   = src[RIDX(i, j, dim)];
            dst[RIDX(dim-1-j, i+1, dim)] = src[RIDX(i+1, j, dim)];
            dst[RIDX(dim-1-j, i+2, dim)] = src[RIDX(i+2, j, dim)];
            dst[RIDX(dim-1-j, i+3, dim)] = src[RIDX(i+3, j, dim)];

            dst[RIDX(dim-1-j-1, i, dim)]   = src[RIDX(i, j+1, dim)];
            dst[RIDX(dim-1-j-1, i+1, dim)] = src[RIDX(i+1, j+1, dim)];
            dst[RIDX(dim-1-j-1, i+2, dim)] = src[RIDX(i+2, j+1, dim)];
            dst[RIDX(dim-1-j-1, i+3, dim)] = src[RIDX(i+3, j+1, dim)];

            dst[RIDX(dim-1-j-2, i, dim)]   = src[RIDX(i, j+2, dim)];
            dst[RIDX(dim-1-j-2, i+1, dim)] = src[RIDX(i+1, j+2, dim)];
            dst[RIDX(dim-1-j-2, i+2, dim)] = src[RIDX(i+2, j+2, dim)];
            dst[RIDX(dim-1-j-2, i+3, dim)] = src[RIDX(i+3, j+2, dim)];

            dst[RIDX(dim-1-j-3, i, dim)]   = src[RIDX(i, j+3, dim)];
            dst[RIDX(dim-1-j-3, i+1, dim)] = src[RIDX(i+1, j+3, dim)];
            dst[RIDX(dim-1-j-3, i+2, dim)] = src[RIDX(i+2, j+3, dim)];
            dst[RIDX(dim-1-j-3, i+3, dim)] = src[RIDX(i+3, j+3, dim)];
        }
```

```
}

pixel src[MAX*MAX];
pixel dst[MAX*MAX];

int main()
{
    int i;
    for (i = 0; i < 20; i++)
        rotate(MAX, src, dst);
    return 0;
}
```

2.4 最后的尝试

矩形分块 32*1，32 路循环展开，并使 dest 地址连续，以减少存储器写次数

```
#include<stdio.h>
#define MAX 4096
#define RIDX(i, j, dim) ((i)*(dim) + (j))
#define COPY(d, s) *(d) = *(s)

typedef struct{
    int red;
    int green;
    int blue;
}pixel;

void rotate(int dim, pixel *src, pixel *dst)
{
    int i, j;
    for (i=0; i < dim; i+=32)
    for (j=dim-1; j >= 0; j-=1) {
        pixel *dptr = dst+RIDX(dim-1-j,i,dim);
        pixel *sptr = src+RIDX(i,j,dim);
        COPY(dptr   , sptr);sptr+=dim; COPY(dptr+1 , sptr);sptr+=dim;
        COPY(dptr+2 , sptr);sptr+=dim; COPY(dptr+3 , sptr);sptr+=dim;
        COPY(dptr+4 , sptr);sptr+=dim; COPY(dptr+5 , sptr);sptr+=dim;
        COPY(dptr+6 , sptr);sptr+=dim; COPY(dptr+7 , sptr);sptr+=dim;
        COPY(dptr+8 , sptr);sptr+=dim; COPY(dptr+9 , sptr);sptr+=dim;
        COPY(dptr+10, sptr);sptr+=dim; COPY(dptr+11, sptr);sptr+=dim;
        COPY(dptr+12, sptr);sptr+=dim; COPY(dptr+13, sptr);sptr+=dim;
        COPY(dptr+14, sptr);sptr+=dim; COPY(dptr+15, sptr);sptr+=dim;
```

```
27          COPY(dptr+16, sptr);sptr+=dim; COPY(dptr+17, sptr);sptr+=dim;
28          COPY(dptr+18, sptr);sptr+=dim; COPY(dptr+19, sptr);sptr+=dim;
29          COPY(dptr+20, sptr);sptr+=dim; COPY(dptr+21, sptr);sptr+=dim;
30          COPY(dptr+22, sptr);sptr+=dim; COPY(dptr+23, sptr);sptr+=dim;
31          COPY(dptr+24, sptr);sptr+=dim; COPY(dptr+25, sptr);sptr+=dim;
32          COPY(dptr+26, sptr);sptr+=dim; COPY(dptr+27, sptr);sptr+=dim;
33          COPY(dptr+28, sptr);sptr+=dim; COPY(dptr+29, sptr);sptr+=dim;
34          COPY(dptr+30, sptr);sptr+=dim; COPY(dptr+31, sptr);
35      }
36  }
37
38  pixel src[MAX*MAX];
39  pixel dst[MAX*MAX];
40
41  int main()
42  {
43      int i;
44      for (i = 0; i < 20; i++)
45          rotate(MAX, src, dst);
46      return 0;
47  }
```

3 实验结果与分析

使用 perf 工具分析程序性能，在具体实现过程中，将图像的两个维度大小定为 4096。同时为了方便观察各版本运行时间以及降低初始化等操作对时间开销的影响，每个版本的代码都将转置操作重复执行 20 次。
在使用 perf 时，选择 task-clock,cycles,instructions,cache-misses,cache-references L1-dcache-load-misses 这几个性能事件，观察不同版本程序之间的区别，实验结果如下：

3.1 原始代码

可以看到，原始代码的运行时间为 5.57 秒，cache miss 次数为 1,176,577,616，L1 cache miss 次数为 906,728,585

3.2 第一次尝试

进行 4*4 数据分块后，发现运行时间从 5.57 秒下降到 1.97 秒。可以发现两者 perf 跑出来的结果主要区别在 cache-misses 和 L1-dcache-load-misses 这两个指标上。改进后的代码这两个指标都比源代码小，说明改进后的代码由于划分了数据块，使得数据更加适应 cpu 的 cache，可以充分利用 cache，提高局部性从而提升性能

```
jyfan@ThinkPad:~/Desktop/HPC$ sudo perf stat -e task-clock,cycles,instructions,cache-misses,cache-references
,L1-dcache-load-misses,node-loads ./a

 Performance counter stats for './a':

          5,563.25 msec task-clock                #    1.000 CPUs utilized
    17,171,102,262      cycles                    #    3.087 GHz                      (66.64%)
    12,294,659,615      instructions              #    0.72  insn per cycle           (83.32%)
     1,174,866,836      cache-misses              #   60.453 % of all cache refs      (83.32%)
     1,943,445,023      cache-references          #  349.336 M/sec                    (83.32%)
       903,977,509      L1-dcache-load-misses                                       (83.36%)
         3,386,916      node-loads                #    0.609 M/sec                    (83.36%)

       5.563691897 seconds time elapsed

       5.491530000 seconds user
       0.071993000 seconds sys
```

图 1: Naive Rotate

```
jyfan@ThinkPad:~/Desktop/HPC$ sudo perf stat -e task-clock,cycles,instructions,cache-misses,cache-references
,L1-dcache-load-misses,node-loads ./b

 Performance counter stats for './b':

          1,971.07 msec task-clock                #    1.000 CPUs utilized
     6,082,910,932      cycles                    #    3.086 GHz                      (66.64%)
    13,840,524,379      instructions              #    2.28  insn per cycle           (83.36%)
       445,274,923      cache-misses              #   70.595 % of all cache refs      (83.36%)
       630,743,007      cache-references          #  320.000 M/sec                    (83.36%)
       216,639,357      L1-dcache-load-misses                                       (83.36%)
           644,339      node-loads                #    0.327 M/sec                    (83.28%)

       1.971398767 seconds time elapsed

       1.895349000 seconds user
       0.075973000 seconds sys
```

图 2: 第一次尝试

3.3 第二次尝试

```
jyfan@ThinkPad:~/Desktop/HPC$ sudo perf stat -e task-clock,cycles,instructions,cache-misses,cache-references
,L1-dcache-load-misses,node-loads ./c

 Performance counter stats for './c':

          1,436.56 msec task-clock                #    1.000 CPUs utilized
     4,432,556,837      cycles                    #    3.086 GHz                      (66.59%)
    11,579,018,425      instructions              #    2.61  insn per cycle           (83.29%)
       343,176,403      cache-misses              #   83.011 % of all cache refs      (83.29%)
       413,410,501      cache-references          #  287.778 M/sec                    (83.29%)
       145,383,865      L1-dcache-load-misses                                       (83.48%)
         1,020,058      node-loads                #    0.710 M/sec                    (83.35%)

       1.436794088 seconds time elapsed

       1.372773000 seconds user
       0.064036000 seconds sys
```

图 3: 第二次尝试

采用 32*32 分块，同时进行 4*4 路循环展开，运行时间降到 1.43 秒。这一步主要有两个方面的优化：数据块划分和循环展开。采用更加合适的数据块划分，进一步优化访存利用局部性减少 cache miss，可以看到 cache miss 次数进一步下降，降低了访存压力。同时对循环进行展开，这样的好处是可以有效发挥处理器指令集并行的能力，提高循环体执行效率，可以看到每周期执行的指令数与上一个版本相比有了略微的提升。

3.4 最后的尝试

```
jyfan@ThinkPad:~/Desktop/HPC$ sudo perf stat -e task-clock,cycles,instructions,cache-misses,cache-references
,L1-dcache-load-misses,node-loads ./d

 Performance counter stats for './d':

          1,113.57 msec task-clock                #    1.000 CPUs utilized
     3,437,213,300      cycles                    #    3.087 GHz                      (66.24%)
     5,160,892,088      instructions              #    1.50  insn per cycle           (83.12%)
       255,927,105      cache-misses              #   88.553 % of all cache refs      (83.46%)
       289,011,671      cache-references          #  259.535 M/sec                    (83.48%)
        78,294,009      L1-dcache-load-misses                                       (83.48%)
           129,017      node-loads                #    0.116 M/sec                    (83.35%)

       1.113973872 seconds time elapsed

       1.029536000 seconds user
       0.083799000 seconds sys
```

图 4: 第三次尝试

矩形分块 32*1，32 路循环展开，并使 dest 地址连续，访存次数进一步减少，通过使 dest 地址连续，定义宏函数拷贝等方式，降低访存压力；通过循环展开提升 CPU 指令级并行的工作效率，最终运行时间降到 1.11 秒