

Tune Performance Domain Specific Language-Halide

wegatron

December 9, 2021

Table of contents

① 性能相关的那些事

② 性能优化的思考

③ DSL-Halide

性能挑战

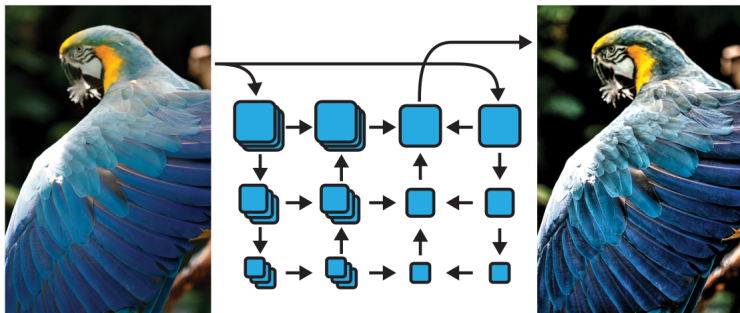


Figure: 在实际应用中, 我们往往需要绞尽脑汁对算法做各种优化, 以满足在移动设备上的流畅运行.

实例分析-初始代码

———— (a) Clean C++ : 9.94 ms per megapixel ————

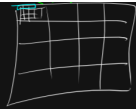
```
void blur(const Image &in, Image &blurred) {  
    Image tmp(in.width(), in.height());  
  
    for (int y = 0; y < in.height(); y++)  
        for (int x = 0; x < in.width(); x++)  
            tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;  
  
    for (int y = 0; y < in.height(); y++)  
        for (int x = 0; x < in.width(); x++)  
            blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;  
}
```

Figure: box filter simple c++ code

inner loops
inside files

9.9 \rightarrow 0.9 ms/megapixel

```
void box_filter_3x3(const Image &in, Image &blurry) {
    __m128i one_third = _mm_set1_epi16(1284);
    #pragma omp parallel for
    for (int yTile = 0; yTile < in.height(); yTile += 32) {
        __m128i a, b, c, sum, avg;
        __m128i blurx[(256/8)*(32+2)]; // allocate tile blurry array
        for (int xTile = 0; xTile < in.width(); xTile += 256) {
            __m128i *blurxPtr = blurx;
            for (int y = 0; y < Y+1; y++) {
                const uint16_t *inPtr = &(in[yTile+y][xTile]);
                for (int x = 0; x < 256; x += 8) {
                    a = _mm_loadu_si128((__m128i*)(inPtr+1));
                    b = _mm_loadu_si128((__m128i*)(inPtr+4));
                    c = _mm_load_si128((__m128i*)(inPtr+7));
                    sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
                    avg = _mm_mulhi_epi16(sum, one_third);
                    _mm_store_si128(blurxPtr++, avg);
                    inPtr += 8;
                }
            }
            blurxPtr = blurx;
            for (int y = 0; y < Y + 32; y++) {
                __m128i *outPtr = (__m128i *)&(blurry[yTile+y][xTile]);
                for (int x = 0; x < 256; x += 8) {
                    a = _mm_load_si128(blurxPtr+(2*256/8));
                    b = _mm_load_si128(blurxPtr+(256/8));
                    c = _mm_load_si128(blurxPtr++);
                    sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
                    avg = _mm_mulhi_epi16(sum, one_third);
                    _mm_store_si128(outPtr++, avg);
                }
            }
        }
    }
}
```



Liles

Tiled, fused
Vectorized
Multithreaded
Redundant
computation
*Near roof-line
optimum*

Figure: 使用矢量计算, tiling, 并行优化之后的快 10 倍的代码

影响因素分析

以出行做类比:

- 算法的设计/参数——计算量 ($O(n)$, $O(n^2)$, ...).
出行的目的地/路线 (抄小道)
- 代码实现. 重复计算/无效计算, 并行, 矢量计算, 内存操作.
出行具体时间的安排 (堵车, 等车...)
- 硬件能力.
出行交通工具 (步行, 骑车, 汽车, 飞机)

优化方向

我们分两方向进行思考:

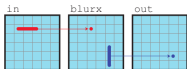
- Algorithm: 如何设计高效的算法 (算法定义了计算量)
- Schedule: 如何将算法编码为高效的机器二进制码

The Schedule

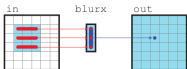
Schedule 包含的内容:

- 遍历的顺序
- 引用的计算是重算还是复用之前结果
- 如何将任务映射到多线程并行, SIMD 指令, GPU

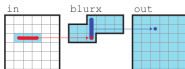
Schedule Example



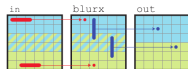
breadth first: each function is entirely evaluated before the next one.



total fusion: values are computed on the fly each time that they are needed.



sliding window: values are computed when needed then stored until not useful anymore.



tiles: overlapping regions are processed in parallel, functions are evaluated one after another.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

serial y, serial x

domain order

1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35
6	12	18	24	30	36

serial x, serial y

1	2
3	4
5	6
7	8
9	10
11	12

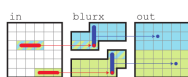
serial y
vectorized x

1	2
1	2
1	2
1	2
1	2
1	2

parallel y
vectorized x

1	2	5	6	9	10
3	4	7	8	11	12
13	14	17	18	21	22
15	16	19	20	23	24
25	26	29	30	33	34
27	28	31	32	35	36

split x into $2x_o + x_i$,
split y into $2y_o + y_i$,
serial y_o, x_o, y_i, x_i



sliding windows within tiles:
tiles are evaluated in parallel
using sliding windows.

Figure: box filter 的几种可能的 Schedule 方案

直接优化代码

- 对程序员有很高的要求.
- 费力耗时
- 且代码与平台相关性高不易移植.

Using Halide

- Schedule 和 Algorithm 分离
- 支持多种平台代码生成 (cpu, metal, opengl, cuda ...)

Halide-Example

———— (c) Halide : 0.90 ms per megapixel ————

```
Func halide_blur(Func in) {  
  Func tmp, blurred;  
  Var x, y, xi, yi;  
  
  // The algorithm  
  tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;  
  blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;  
  
  // The schedule  
  blurred.tile(x, y, xi, yi, 256, 32)  
    .vectorize(xi, 8).parallel(y);  
  tmp.chunk(x).vectorize(x, 8);  
  
  return blurred;  
}
```

Figure: halide box filter 代码实现

自动调优

Schedule 独立, 方便了性能调优, 但任然需要耗时费力的尝试.
有没有办法, 自动得到一个好的方案?

解空间的建模

Schedule 变化可以总结为两个方面:

- The Domain Order, 定义了每个函数空间的遍历方式
 - 顺序/并行
 - 循环展开以及矢量化
 - column major/row major
 - 拆分为多个小片, 按片遍历
- The Call Schedule, 定义了函数存储和计算的粒度如:
 - breadth-first, 先计算 blur-x 存储起来, 然后再计算 blur
 - fused, 每次重新计算 blur-x
 - sliding-window

解空间的搜索

变化的空间非常大, 使用遗传算法, 并通过一些先验来搜索. 一些策略:

- 从 breadth-first 开始进行搜索
- 片的长度的选择为 2 的整数倍
- 同一个函数采用相同的策略
- ...

Compiler Pipeline

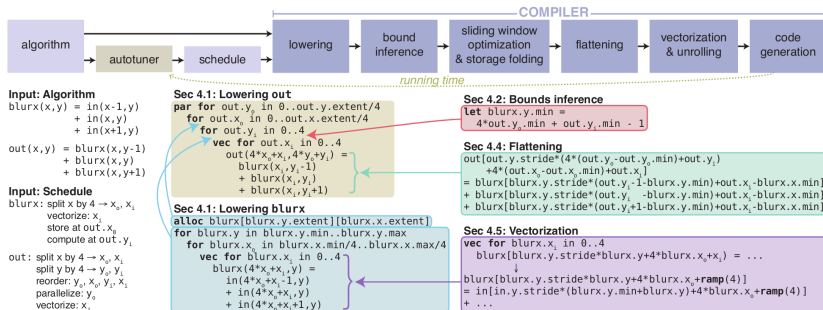


Figure: 编译流程

Reference

- Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines
- Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines