并行 指令 数据

并行

» 其实没有什么太大的优化空间

» 优化第一次筛选？

*$\sqrt{10^9}$以下的质数也只有3401个

· 欧拉筛？打表？数据太小没明显区别

*时间复杂度$O(n)$的质数筛法

» 用欧拉筛筛第二次？（要是欧拉筛能并行就好了）

· 接近随机的内存访问

指令

» 指令数量

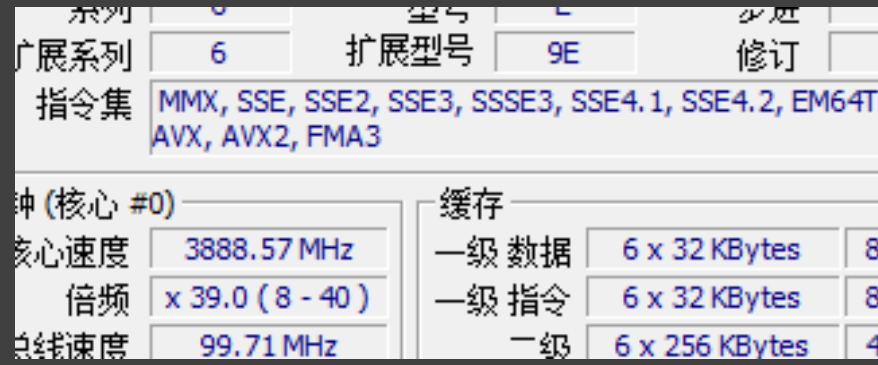
· 程序本身的代码简化

» 单指令以及指令组合的效率

```
c = b * (1 - ((a & 1) << 1)); // 快
c = (a & 1) ? -b : b; // 慢
```

» 单指令吞吐率

· SIMD，指令级并行



数据

» 减少访存次数

· 预处理生成查询表，以空间换时间

» 提高局部性

· Cache性能的考虑

· 集中访存

» 统一类型

· 避免数据在CPU中的长度/类型转换

1

压缩标记表*marked*

» 每*int*8标记一个数→每*bit*标记一个数

```
marked[(num - lBound) / 2] = true;

int index = (num - lBound) >> 1;
marked[index >> 3] |= 1 << (index & 0b111);
```

» 虽然这样做指令数翻了个倍

但是访存的跨度只有原来的1/8！

2

预计算$COUNT_BITS$表

» 压缩$marked$表后统计质数个数：

```
int count = totalNumsInTheBlock;
for (int i = 0; i < marked.size(); i++)
    for (int j = 0; j < 8; j++)
        count -= !(~marked[i] & (1 << j))
```

» 使用$COUNT_BITS$查表优化：

```
int count = totalNumsInTheBlock;
for (int i = 0; i < marked.size(); i++)
    count -= COUNT_BITS[marked[i]];
```

» 统计的时间降低至1/8

» 更激进，生成16位的表（大小为$2^{16} \times 1B = 64KB$，可以接受）

3

SIMD指令（*AVX2*）

» 16个256位寄存器

» 下标需要*int*32来存储，一次可以算8个下标

```
int stride = prime * 2;

// SISD calculation
for (int i = start; i <= end; i += stride) mark(i);

// SIMD vectorized calculation
int times = ceil((end - start) / (stride * 8));
Vector8 vreg1 = Vector8(start) + Vector8(0, 1, 2, 3, .., 7) * stride;
for (int i = 0; i < times; i++)
{
    Vector8 vreg2 = calcIndices(vreg1);
    int indices[8];
    store(vreg2, indices);
    for (int j = 0; j < 8; j++) mark(indices[j]);
    vreg1 += Vector8(stride * 8);
}
```

3

SIMD指令（*AVX2*）

» 如何使用？*intrinsic*库<immintrin.h>

```
pindex = _mm256_sub_epi32(poff, P_LVAL);
pindex = _mm256_srli_epi32(pindex, 1);
pbits = _mm256_and_si256(pindex, P_0X7);
pbits = _mm256_sllv_epi32(P_ONE, pbits);
pindex = _mm256_srli_epi32(pindex, 3);

_mm256_storeu_si256((__m256i*)indices, pindex);
_mm256_storeu_si256((__m256i*)bits, pbits);

for (int k = 0; k < 8; k++) marked[indices[k]] |
```

*需要打开-mavx -march=native编译选项

» 开-O以上编译才会有优化效果，不然：

```
1067    movl    1176(%rbp), %eax
1068    cmpl    1180(%rbp), %eax
1069    jge .L79
1070    vmovdqa 768(%rbx), %ymm0
1071    vmovdqa %ymm0, 352(%rbx)
1072    vmovdqa 736(%rbx), %ymm0
1073    vmovdqa %ymm0, 320(%rbx)
1074    vmovdqa 352(%rbx), %ymm0
1075    vmovdqa 320(%rbx), %ymm1
1076    vpsubd  %ymm1, %ymm0, %ymm0
1077    vmovdqa %ymm0, 576(%rbx)
1078    vmovdqa 576(%rbx), %ymm0
1079    vmovdqa %ymm0, 384(%rbx)
1080    movl    $1, 980(%rbp)
1081    vmovdqa 384(%rbx), %ymm0
1082    vmovd   980(%rbp), %xmm1
1083    vpsrld  %xmm1, %ymm0, %ymm0
```

```
1083    vpsrld  %xmm1, %ymm0, %ymm0
1084    vmovdqa %ymm0, 576(%rbx)
1085    vmovdqa 576(%rbx), %ymm0
1086    vmovdqa %ymm0, 224(%rbx)
1087    vmovdqa 704(%rbx), %ymm0
1088    vmovdqa %ymm0, 192(%rbx)
1089    vmovdqa 224(%rbx), %ymm1
1090    vmovdqa 192(%rbx), %ymm0
1091    vpand   %ymm0, %ymm1, %ymm0
1092    vmovdqa %ymm0, 544(%rbx)
1093    vmovdqa 672(%rbx), %ymm0
1094    vmovdqa %ymm0, 288(%rbx)
1095    vmovdqa 544(%rbx), %ymm0
1096    vmovdqa %ymm0, 256(%rbx)
1097    vmovdqa 256(%rbx), %ymm1
1098    vmovdqa 288(%rbx), %ymm0
1099    vpsllvd %ymm1, %ymm0, %ymm0
```

*最终的测试不开任何编译选项

3

SIMD指令（$AVX2$）

» 想在不开任何$flag$编译的情况下用到?

· 手动写GCC内联汇编!

```
asmv("vpbroadcastd  (%0), %%ymm4" : : "r"(&dstride));   // pstride = dstride
asmv("vpmulld       %ymm4, %ymm3, %ymm5");              // poff = pscale * pstride
asmv("vpbroadcastd  (%0), %%ymm8" : : "r"(&start));     // ymm8 = start
asmv("vpaddd        %ymm5, %ymm8, %ymm5");              // poff = ymm8 + poff = start + pscale * dstride
asmv("vpslld        $3, %ymm4, %ymm4");                 // pstride <<= 3

int indices[8], bits[8];

for (int j = 0; j < times; j++)
{
    asmv("vpsubd    %ymm0, %ymm5, %ymm6");  // pindex = poff - lVal
    asmv("vpsrld    $1, %ymm6, %ymm6");     // pindex >>= 1
    asmv("vpand     %ymm1, %ymm6, %ymm7");  // pbits = pindex & 0x7
    asmv("vpsllvd   %ymm7, %ymm2, %ymm7");  // pbits = 1 << pbits
    asmv("vpsrld    $3, %ymm6, %ymm6");     // pindex >>= 3

    asmv("vmovdqa   %%ymm6, %0" : "=m"(indices));// store pindex -> indices
    asmv("vmovdqa   %%ymm7, %0" : "=m"(bits));  // store pbits -> bits

    for (int k = 0; k < 8; k++) marked[indices[k]] |= bits[k];
    asmv("vpaddd    %ymm4, %ymm5, %ymm5"); // poff += pstride
}
```

4

利用编译器的智慧

» 开-O3编译

» 再反编译看哪里被优化了

```
for (int k = 0; k < 8; k++) marked[indices[k]] |= bits[k];
asmv("vpaddd    %ymm4, %ymm5, %ymm5"); // poff += pstride
```

```
*(_BYTE *)(v23 + indices[0]) |= LOBYTE(bits[0]);
*(_BYTE *)(v23 + indices[1]) |= LOBYTE(bits[1]);
*(_BYTE *)(v23 + indices[2]) |= LOBYTE(bits[2]);
*(_BYTE *)(v23 + indices[3]) |= LOBYTE(bits[3]);
*(_BYTE *)(v23 + indices[4]) |= LOBYTE(bits[4]);
*(_BYTE *)(v23 + indices[5]) |= LOBYTE(bits[5]);
*(_BYTE *)(v23 + indices[6]) |= LOBYTE(bits[6]);
*(_BYTE *)(v23 + indices[7]) |= LOBYTE(bits[7]);
__asm { vpaddd  ymm5, ymm5, ymm4 }
```

» 省去循环的8次分支跳转，2.1s → 1.3s

» 去除LOBYTE转换，1.3s → 1.2s

*

Hack/Trick

» `#pragma GCC optimize("3")`

» 只求个数，不关心具体的质数是哪些

- Meissel-Lehmer algorithm $O\left(\frac{n^{\frac{2}{3}}}{logn}\right)$