

N方过百万，暴力踩标算 ——神奇的指令集

成都市第七中学 王思齐 袁方舟



从流传在OI界的一条神秘语句说起...

- ▶ 想必大家已经在很多地方见到过这条语句了。。。

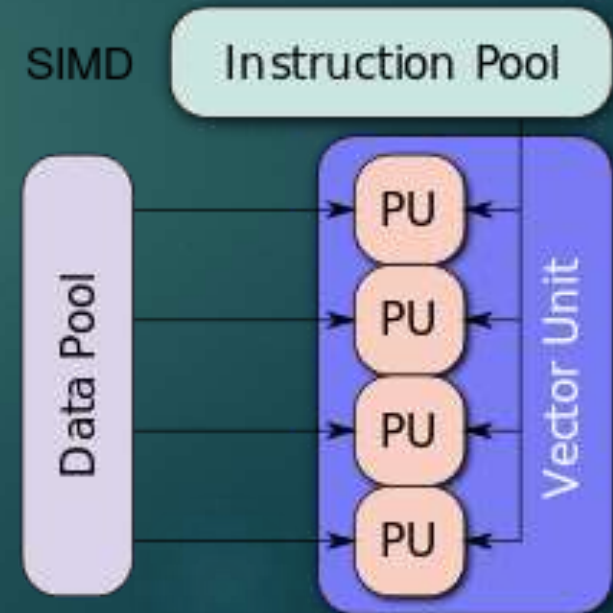
```
#pragma GCC optimize("Ofast,no-stack-protector,unroll-loops,fast-math")  
#pragma GCC target("sse,sse2,sse3,ssse3,sse4.1,sse4.2,avx,avx2,popcnt,tune=native")
```

- ▶ 其中#pragma GCC target就是在指定目标指令集。

- ▶ 可惜的是，在大多数时候，光有这条语句还不够，还得手动使用相关指令才行。

什么是指令集？有什么用？

- ▶ 指令集架构，包含一系列的opcode即操作码（机器语言），以及由特定处理器执行的基本命令。常见的指令集架构有Intel和AMD搞的x86指令集架构，ARM以及MIPS指令集架构（比如我国的龙芯就是类似MIPS架构的）。
- ▶ 接下来所讲的指令集，指的是x86架构上的SIMD指令集。即“单指令多数据 (Single Instruction Multiple Data)”技术。顾名思义，就是用一条指令快速操作一组数据。
- ▶ SIMD指令集有许多，这里只讲一部分
- ▶ 至于它有什么用？当然是快，快，快，越快越好啦！



这些指令集都是些什么？

```
#pragma GCC target("sse,sse2,sse3,ssse3,sse4.1,sse4.2,avx,avx2,popcnt,tune=native")
```

- ▶ 可以看到，我们使用了"SSE,SSE2,SSE3,SSE4.1,SSE4.2,SSSE3,avx,avx2,popcnt"这些指令集。他们都是些什么呢？
- ▶ SSE(Streaming SIMD Extensions)，是x86架构处理器中的指令集。这个指令集最先由Intel推出（然后在Intel和AMD的战争中发扬光大），SSE2,SSE3,SSE4.1,SSE4.2,SSSE3均是该指令集的拓展。这个指令集使用了8个xmm寄存器来执行128位运算。
- ▶ 而AVX,AVX2(Advanced Vector eXtensions)，是SSE的升级版。这个指令集拓展了xmm寄存器（被命名成ymm寄存器），使得能进行的运算从128位变成了256位。
- ▶ popcnt，就是计算popcount的东西。该指令集使得一条指令就能计算popcount。

如何使用它们？

- ▶ 万幸的是，Intel提供了一些头文件（如immintrin.h, emmintrin.h等），这些头文件把指令用函数的方式进行封装（而GCC使用了内建函数来实现），这使得我们不需要写内联汇编来调用指令集中的指令。只需要使用相关的数据类型，我们就能完成操作。

```
extern __inline __m256 __attribute__((__gnu_inline__, __always_inline__, __artificial__))
_mm256_div_ps (__m256 __A, __m256 __B)
{
    return (__m256) __builtin_ia32_divps256 ((__v8sf)__A, (__v8sf)__B);
}
```

- ▶ 事不宜迟，让我们来见识一下如何使用它来对抗毒瘤的出题人吧！

暴力过题？

- ▶ (一道来自Ynoi2018的题“五彩斑斓的世界”)
- ▶ 二阶堂真红给了你一个长为 n 的序列 a ，有 m 次操作
- ▶ 1.把区间 $[l,r]$ 中大于 x 的数减去 x
- ▶ 2.查询区间 $[l,r]$ 中 x 的出现次数
- ▶ 所有输入的数均在 $[0,100000]$

暴力过题？

- ▶ 想必大家都可以随手写出一个 $O(nm)$ 暴力（当然，你要是说随手写 $O(n\sqrt{n})$ 也行）
- ▶ 那么这道题如何使用指令集呢？
- ▶ 可以发现，对于32位整数，AVX2为我们提供了8个数一起减，8个数一起比较，256个位一起与等操作。

暴力过题？

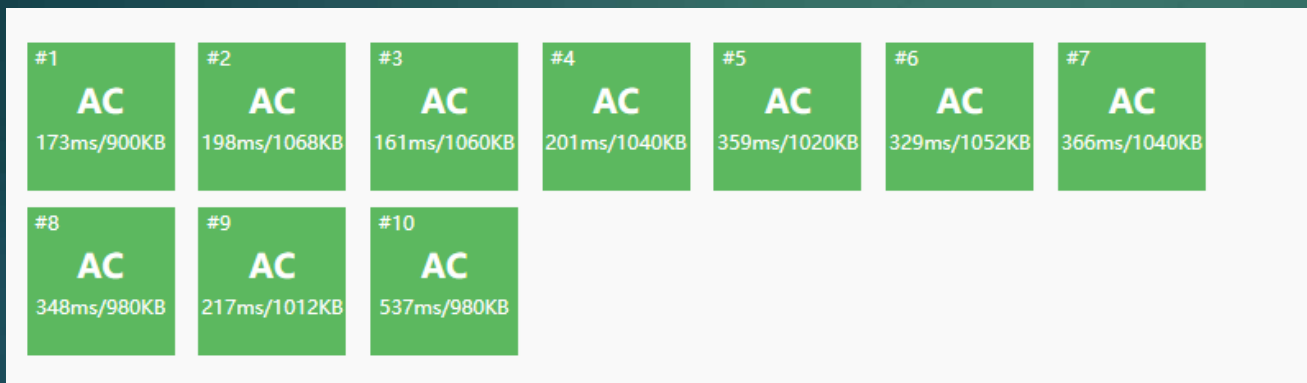
- ▶ 这就是使用指令集的暴力代码
- ▶ 为了处理方便，我们需要让数据对齐。
- ▶ 那么我们就可以做到 $O(nm/8)$ 的复杂度！它的实际效果怎么样呢？

```
inline void modify(int l,int r,int x)
{
    r++;
    while((l&7)&&1<r)s[l]>x?s[l]-=x:0,l++;
    if(l==r)return;
    while(r&7)r--,s[r]>x?s[r]-=x:0;
    if(l==r)return;
    l>>=3,r>>=3;
    __m256i*s=_s+1,t=_mm256_set_epi32(x,x,x,x,x,x,x,x);
    for(r-=1;r;r--,s++)*s=_mm256_sub_epi32(*s,_mm256_and_si256(_mm256_cmpgt_epi32(*s,t),t));
}

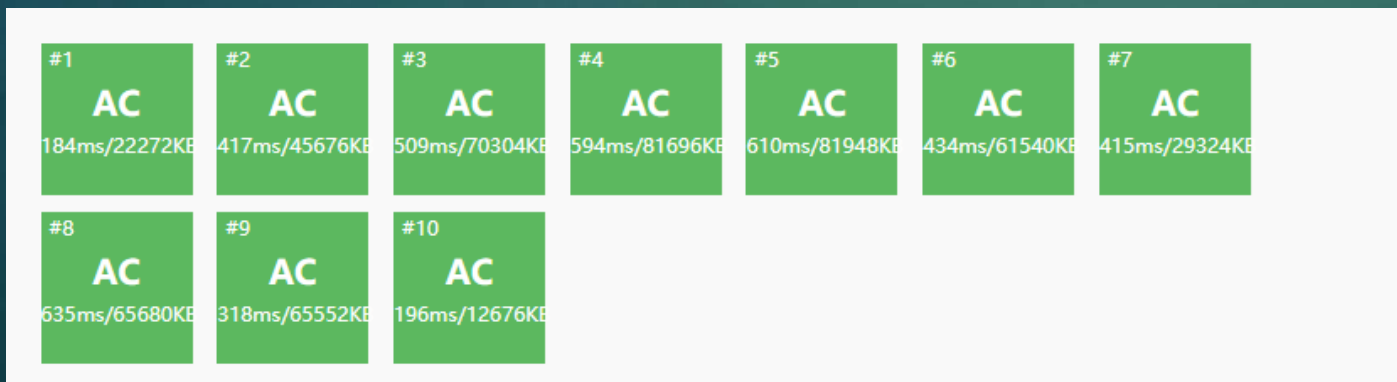
inline void query(int l,int r,int x)
{
    r++;
    while((l&7)&&1<r)q+=s[l++]==x;
    if(l==r)return;
    while(r&7)q+=s[--r]==x;
    if(l==r)return;
    l>>=3,r>>=3;
    __m256i*s=_s+1,t=_mm256_set_epi32(x,x,x,x,x,x,x,x),res=_mm256_setzero_si256();
    for(r-=1;r;r--,s++)res=_mm256_add_epi32(res,_mm256_cmpeq_epi32(*s,t));
    fo0(i,8)q-=((int*)&res)[i];
}
```


暴力过题？

- ▶ 这是使用指令集的暴力程序（总用时：2889ms / 内存：1068KB）
- ▶ （卡内存？不存在的！）



- ▶ 这是lxl的标程（总用时: 4312ms / 内存: 81948KB）



为什么这么快？

- ▶ 本题大部分数据不是针对 $O(nm)$ 暴力造的，所以会造成总用时差别很大的假象
- ▶ $1e9$ 条指令在计算机上其实非常快。可以观察到我们使用的指令时钟周期都很少。
- ▶ 可惜的是，虽然除了常数，但是复杂度还是 $O(nm)$ 。这导致如果 n,m 扩倍的话时间增长和标算差得远。比如 $n,m=3e5$ 的时候指令集暴力就会原地爆炸，会比 $n,m=1e5$ 慢整整9倍。。。

暴力过题II?

- ▶ 这道题来自2018年集训队作业，UOJ#435 Simple Tree

有一棵有根树，根为 1，点有点权。

现在有 m 次操作，操作有 3 种：

1 $x y w$ ，将 x 到 y 的路径上的点点权加上 w (其中 $w = \pm 1$)；

2 $x y$ ，询问在 x 到 y 的路径上有多少个点点权 > 0 ；

3 x ，询问在 x 的子树里的点有多少个点点权 > 0 。

- ▶ 数据范围， $n, m \leq 100000$

暴力过题II?

- ▶ 树剖后，用 $O(\log)$ 的代价转成若干个不相交区间的操作，暴力复杂度为 $O(m\log n + mn)$
- ▶ 同样的，利用AVX2给我们提供的“8个数一起加，8个数一起比较”，可以做到 $O(nm/8)$ 。而它的效果也是惊人的（比我辛辛苦苦写的正解还快呜呜呜(当然实际上没那么快)）

#305662	#435. 【集训队作业2018】 Simple Tree	kczno1	100	23157ms	23172kb	C++11	4.5kb	2018-12-11 15:55:23
#305483	#435. 【集训队作业2018】 Simple Tree	std	100	23528ms	22180kb	C++	4.3kb	2018-12-10 22:48:55
#305513	#435. 【集训队作业2018】 Simple Tree	wangxiuhan	100	26358ms	51708kb	C++11	8.9kb	2018-12-11 09:05:27
#305726	#435. 【集训队作业2018】 Simple Tree	mcfxmcfx	100	26659ms	56728kb	C++11	8.4kb	2018-12-11 20:24:22
#307822	#435. 【集训队作业2018】 Simple Tree	negiizhao	100	26674ms	67548kb	C++11	7.6kb	2018-12-22 01:24:36
#305617	#435. 【集训队作业2018】 Simple Tree	yww	100	27190ms	29616kb	C++11	6.7kb	2018-12-11 14:06:13
#305520	#435. 【集训队作业2018】 Simple Tree	yfzsc	100	27955ms	33152kb	C++11	6.0kb	2018-12-11 09:27:00

想叉掉？对不起，叉不掉！

- ▶ 对于正解的分块程序，块大小很玄学，常数写得不好容易被对着卡
- ▶ 虽然可以很方便地让指令集暴力跑满，然而指令集就算跑满也是那个速度。且通常使用指令集暴力的程序内存都非常小，访问内存也连续。
- ▶ 得出结论：虽然可以卡满，但是只有一种方法卡满，这种方法还卡不掉...

#7899	#305660	#435. 【集训队作业2018】 Simple Tree	fjzzq2002	mcfxmcfx	Failed.	2018-12-11 16:35:15
#7898	#305660	#435. 【集训队作业2018】 Simple Tree	fjzzq2002	mcfxmcfx	Failed.	2018-12-11 16:29:19
#7897	#305660	#435. 【集训队作业2018】 Simple Tree	fjzzq2002	mcfxmcfx	Failed.	2018-12-11 16:26:43
#7895	#305660	#435. 【集训队作业2018】 Simple Tree	fjzzq2002	mcfxmcfx	Failed.	2018-12-11 16:24:52

__m256i是什么

- ▶ `m256i`是编译器内置的一个类型。GCC中，他们的定义在头文件里是这样的。SSE的 `m128i`定义也差不多。

```
typedef float  __m256  __attribute__((__vector_size__(32),  
                                     __may_alias__));  
typedef long long __m256i __attribute__((__vector_size__(32),  
                                     __may_alias__));  
typedef double __m256d __attribute__((__vector_size__(32),  
                                     __may_alias__));
```

- ▶ 其中，“long long”指定了每一个“单位”的大小，`__vector_size__(X)`指定了总大小为X字节。这个X必须是2的幂，而且X必须是“单位”的倍数。`__may_alias__`是一个奇怪的选项，具体见<https://gcc.gnu.org/onlinedocs/gcc-5.2.0/gcc/Type-Attributes.html>

Elephant类型

- ▶ 所以，你甚至可以定义一个“Elephant”类型！

```
typedef int __elep __attribute__((__vector_size__(4)))  
__elep a,b,c;
```

- ▶ 然而，这样做的代价是...
- ▶ 沙雕编译器把几千条语句全写出来了！

```
151 0x0000000000471a45 <+981>: movdqa 0xa50(%rbx),%xmm0  
152 0x0000000000471a4d <+989>: paddd 0x650(%rbx),%xmm0  
153 0x0000000000471a55 <+997>: movaps %xmm0,0x1f0(%rbp)  
154 0x0000000000471a5c <+1004>: movdqa 0xa60(%rbx),%xmm0  
155 0x0000000000471a64 <+1012>: paddd 0x660(%rbx),%xmm0  
156 0x0000000000471a6c <+1020>: movaps %xmm0,0x200(%rbp)  
157 0x0000000000471a73 <+1027>: movdqa 0xa70(%rbx),%xmm0  
158 0x0000000000471a7b <+1035>: paddd 0x670(%rbx),%xmm0  
159 0x0000000000471a83 <+1043>: movaps %xmm0,0x210(%rbp)  
160 0x0000000000471a8a <+1050>: movdqa 0xa80(%rbx),%xmm0  
161 0x0000000000471a92 <+1058>: paddd 0x680(%rbx),%xmm0  
162 0x0000000000471a9a <+1066>: movaps %xmm0,0x220(%rbp)  
163 0x0000000000471aa1 <+1073>: movdqa 0xa90(%rbx),%xmm0  
164 0x0000000000471aa9 <+1081>: paddd 0x690(%rbx),%xmm0  
165 0x0000000000471ab1 <+1089>: movaps %xmm0,0x230(%rbp)  
166 0x0000000000471ab8 <+1096>: movdqa 0xaa0(%rbx),%xmm0  
167 0x0000000000471ac0 <+1104>: paddd 0x6a0(%rbx),%xmm0  
168 0x0000000000471ac8 <+1112>: movaps %xmm0,0x240(%rbp)  
169 0x0000000000471acf <+1119>: movdqa 0xab0(%rbx),%xmm0  
170 0x0000000000471ad7 <+1127>: paddd 0x6b0(%rbx),%xmm0  
171 0x0000000000471adf <+1135>: movaps %xmm0,0x250(%rbp)  
172 0x0000000000471ae6 <+1142>: movdqa 0xac0(%rbx),%xmm0  
173 0x0000000000471aee <+1150>: paddd 0x6c0(%rbx),%xmm0  
174 0x0000000000471af6 <+1158>: movaps %xmm0,0x260(%rbp)  
175 0x0000000000471afd <+1165>: movdqa 0xad0(%rbx),%xmm0
```

如何取出第i位？

- ▶ 可以强转指针，也可以使用[]运算符。
- ▶ 聪明的橡树们告诉沙雕的编译器这个类似于一个数组，所以你可以使用如下的操作：

```
__m128i a={1,2};  
printf("[%d]",a[0]);
```


支持四则运算？

- ▶ 聪明的橡树们为这个沙雕编译器特判了各种操作。这些操作有加减乘除，等于和一些位运算。
- ▶ 可惜的是，沙雕编译器对于橡树们没有特判的情况（比如整数除），它就会每个都调用一下普通指令（比如`idiv`）。所以一般还是手动调用函数比较好。
- ▶ 还有一点，虽然写作`a=b+c`，但是沙雕编译器眼里的却是：

```
for(int i=0;i<8;++i)
    a[i]=(b[i]+c[i])%(1<<32);
```

“等于”号的奥秘

- ▶ 使用普通__m256i类型，编译器会使用“vmovaps”指令等来实现内存拷贝
- ▶ 如果使用Elephant类型，编译器有可能会使用一堆mov指令。如果使用很大的Elephant类型，比如最大的8192，这个时候编译器会使用一个“rep movsq”指令。“rep movsq”指令能使CPU满开。它的速度比每次拷贝一个__m256i的速度慢一些，不过比普通的循环快得多。

```
0x0000000004716f6 <+134>:   rep movsq %ds:(%rsi),%es:(%rdi)
```

压64位的bitset？现在流行压256位了！

- ▶ AVX, AVX2指令集给出了许多直接操作256位的指令，这使得bitset可以压更多的位数

<code>__m256i _mm256_and_si256 (__m256i a, __m256i b)</code>	<code>vpand</code>
<code>__m256i _mm256_andnot_si256 (__m256i a, __m256i b)</code>	<code>vpandn</code>
<code>__m256i _mm_broadcastsi128_si256 (__m128i a)</code>	<code>vbroadcasti128</code>
<code>__m256i _mm256_broadcastsi128_si256 (__m128i a)</code>	<code>vbroadcasti128</code>
<code>__m128i _mm256_extracti128_si256 (__m256i a, const int imm8)</code>	<code>vextracti128</code>
<code>__m256i _mm256_inserti128_si256 (__m256i a, __m128i b, const int imm8)</code>	<code>vinseri128</code>
<code>__m256i _mm256_or_si256 (__m256i a, __m256i b)</code>	<code>vpor</code>
<code>__m256i _mm256_permute2x128_si256 (__m256i a, __m256i b, const int imm8)</code>	<code>vperm2i128</code>
<code>__m256i _mm256_slli_si256 (__m256i a, const int imm8)</code>	<code>vpslldq</code>
<code>__m256i _mm256_srli_si256 (__m256i a, const int imm8)</code>	<code>vpsrldq</code>
<code>__m256i _mm256_stream_load_si256 (__m256i const* mem_addr)</code>	<code>vmovntdqa</code>
<code>__m256i _mm256_xor_si256 (__m256i a, __m256i b)</code>	<code>vpxor</code>
<code>int _mm256_testc_si256 (__m256i a, __m256i b)</code>	<code>vptest</code>
<code>int _mm256_testnzc_si256 (__m256i a, __m256i b)</code>	<code>vptest</code>
<code>int _mm256_testz_si256 (__m256i a, __m256i b)</code>	<code>vptest</code>
<code>__m256i _mm256_undefined_si256 (void)</code>	
<code>__m256i _mm256_zextsi128_si256 (__m128i a)</code>	

256位的popcount

- ▶ 可惜的是，POPCNT指令集只支持了64位的popcount快速计算。不过256位会在Intel的AVX-512支持。
- ▶ 不过popcount同样可以快速实现，只是这个实现有点烦。在一篇论文“Faster Population Counts Using AVX2 Instructions”有讲到并给出测速

TABLE 3. Number of cycles per 64-bit input word required to compute the population of arrays of various sizes.

array size	WWG	Lauradoux	HS	popcnt	AVX2 Mula	AVX2 HS
256 B	6.00	4.50	3.25	1.12	1.38	—
512 B	5.56	2.88	2.88	1.06	0.94	—
1 kB	5.38	3.62	2.66	1.03	0.81	0.69
2 kB	5.30	3.45	2.55	1.01	0.73	0.61
4 kB	5.24	3.41	2.53	1.01	0.70	0.54
8 kB	5.24	3.36	2.42	1.01	0.69	0.52
16 kB	5.22	3.36	2.40	1.01	0.69	0.52
32 kB	5.23	3.34	2.40	1.01	0.69	0.52
64 kB	5.22	3.34	2.40	1.01	0.69	0.52

FFT, NTT, 矩阵乘法也可以用指令集？

- ▶ 对于若干个内存连续变量，即可以方便地用指令集来批量处理。我们观察FFT中同样有这样的性质。对于该代码的简单实现，可以去查看UOJ#34的rank1的代码。使用SSE优化，对于相同长度的FFT，用时是普通的1/2。

time: 5339ms

memory: 590380kb

time: 2485ms

memory: 459772kb

- ▶ 对于NTT，如果是不定模数，可以用double的批量除指令优化。
- ▶ 矩阵乘法的形式是 $C(i,j)=\sum A(i,k)*B(k,j)$ ，将B转置后也可以使用指令集处理。
- ▶ 一句话，能够方便地批量处理的玩意都能指令集。内存连续最好。

编译器自动优化

- ▶ 对于一些“编译器友好”的代码，编译器会帮你转成指令集。当然，不开启O3及O3以上的优化或者不开启-mavx2编译器是不会帮你的。
- ▶ 考虑max卷积问题。下图中有两份代码

```
#include <bits/stdc++.h>
using namespace std;

const int n = (1 << 17);
int a[n], b[n], c[n];

int main() {
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    for (int i = 0; i < n; i++) scanf("%d", &b[i]);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            c[i] = max(c[i], a[j] + b[i - j]);
        }
        printf("%d ", c[i]);
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;

const int n = (1 << 17);
int a[n], b[n];

int main() {
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    for (int i = 0; i < n; i++) scanf("%d", &b[i]);
    for (int i = 0; i < n; i++) {
        int c = 0;
        for (int j = 0; j <= i; j++) {
            c = max(c, a[j] + b[i - j]);
        }
        printf("%d ", c);
    }
    return 0;
}
```

编译器自动优化

- ▶ 咋一看几乎完全一样。使用GCC4.8.4编译后，效果则大相径庭：第一份跑了7.3s，第二份跑了1.8s。观察汇编代码得出的结果也是一样：第二份被编译器用指令集+循环展开优化了。
- ▶ 然而，用其他版本GCC编译时（甚至是高版本GCC），第一份代码却是有时候被优化，有时候不被优化。。。
- ▶ 所以。。。
- ▶ 得出的结论是。。。
- ▶ 膜神树？



为什么会RE

- ▶ 如果你使用了某个指令集指令，然而电脑不支持的话，你就会收到一个 SIGILL,illegal instruction 错误。
- ▶ 这样的话可能有点惨（捂脸）

```
-----  
Process exited after 4.7 seconds with return value 3221225501  
请按任意键继续. . . ■
```


和循环展开一起用？

- ▶ 和循环展开一起用是有效的。循环展开的用途是为具有多个功能单元的处理
器提供指令级并行，也有利于指令流水线的调度。
- ▶ 如果你开启了O3或者Ofast，或者是开启了-funroll-loops开关，编译器就
会帮你循环展开。
- ▶ 这个时候如果手动循环展开，按理是会快一点，但实际差别不大。

各大OJ的支持情况

- ▶ bzoj：由于cpu过于古老，只能支持到大约sse2（就是说256位的基本别想用，128位的都有一些用不了）。
- ▶ loj：支持不超过avx的指令集。
- ▶ 评测鸭：理论上可以使用avx2指令集（松松松表示在更新了JudgeduckOS后，SSE都没法使用，但是他会尽快更新以支持指令集）。
- ▶ luogu、hdu：支持avx2。
- ▶ 牛客：理论上支持到avx，但是实测不知为何，可以使用vaddpd，不能使用vmaxpd
- ▶ UOJ和CF的情况有点特殊，__m256i没有定义，但是可以用汇编调用那些指令。
- ▶ 在问过神奇橡树后，我们得到了在UOJ便捷的使用avx2的方法——

在UOJ使用avx2

```
#define __AVX__ 1
#define __AVX2__ 1
#define __SSE__ 1
#define __SSE2__ 1
#define __SSE2_MATH__ 1
#define __SSE3__ 1
#define __SSE4_1__ 1
#define __SSE4_2__ 1
#define __SSE_MATH__ 1
#define __SSSE3__ 1
```

- ▶ 在最前面粘贴这段代码，你会发现，所有指令集都能用了。
- ▶ 原理大概是-mavx2会定义一个宏，头文件里判断是否支持会涉及这个宏。
- ▶ 在开C++11时，bits/stdc++.h中会包含random，而random会包含x86intrin.h，所以这段代码必须在这之前。而C++98中只需要在immintrin.h之前。
- ▶ 这段代码在CF同样可用，但是某些指令无法使用（如_mm256_extract_epi64），这是因为CF是32位的。如果强行钦定成64位，反而会有更多问题出现。

更牛逼的指令集：AVX-512

- ▶ AVX-512是Intel最新推出的指令集。在这个指令集里，Intel把ymm寄存器再次拓展，变成了zmm寄存器（好奇AVX-1024该叫什么寄存器了）
- ▶ 在AVX-512里众多指令得到了增强，还有新指令的出现。当前支持AVX-512的CPU寥寥无几（大概现有的OJ都不会去换个这种CPU吧），有兴趣可以去了解一下（说不定等ccf上了c++11和64位之后就全员支持了）



An advertisement for Intel AVX-512. The background shows a server room with blue lighting. The text at the top reads "英特尔® 高级矢量扩展 512 (英特尔® AVX-512)". The main headline is "加速计算密集型工作负载". Below the headline, it says "英特尔® 高级矢量扩展 512 (英特尔® AVX-512)". At the bottom, there is a paragraph of text: "各个行业领域对更高计算性能的需求仍在不断增加。为了支持日益增长的需求和不断演变的使用模式，我们利用可在最新英特尔® 至强® 处理器和协处理器¹及英特尔® 至强® 可扩展处理器上使用的英特尔® AVX-512 来继续进行工作负载优化创新。"

循环并行化：OpenMP

- ▶ OpenMP是一个多线程编译处理方案。如果在一个循环前面加上`#pragma omp ...`，编译器就会自动帮你多线程执行这段代码。
- ▶ 其实也可以使用pthread实现多线程，但是OpenMP更加方便。
- ▶ 可惜的是，必须在编译选项内加入`-fopenmp`才能开启这个功能，而且算法竞赛的OJ是禁多线程的，使用这个就会得到Dangerous Syscalls（捂脸）

一个神奇的网站(万恶之源)

- ▶ 这个网站叫做：<https://software.intel.com/sites/landingpage/IntrinsicsGuide>，在这个网站里可以搜索需要的函数，以及这些函数所对应指令、所需的指令集、头文件等



The Intel Intrinsics Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code. ✕

Technologies

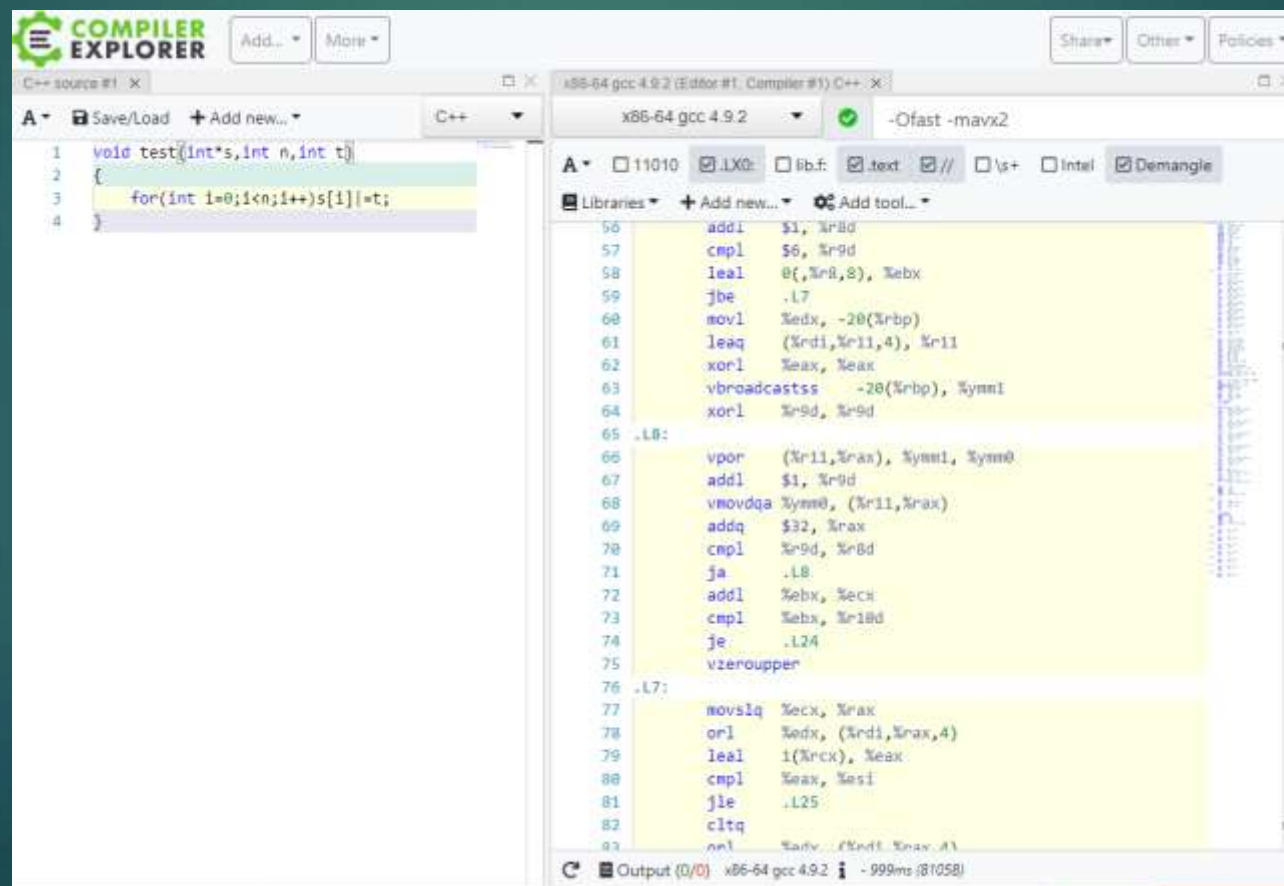
- MMX
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2
- AVX
- AVX2
- FMA
- AVX-512
- KNC
- SVML
- Other

_mm_search

```
__m512i _mm512_4dpwssd_epi32 (_m512i src, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b) vp4dpwssd
__m512i _mm512_mask_4dpwssd_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2,
_m512i a3, _m128i * b) vp4dpwssd
__m512i _mm512_maskz_4dpwssd_epi32 (_mmask16 k, _m512i src, _m512i a0, _m512i a1, _m512i a2,
_m512i a3, _m128i * b) vp4dpwssd
__m512i _mm512_4dpwssds_epi32 (_m512i src, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i *
b) vp4dpwssds
__m512i _mm512_mask_4dpwssds_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2,
_m512i a3, _m128i * b) vp4dpwssds
__m512i _mm512_maskz_4dpwssds_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2,
_m512i a3, _m128i * b) vp4dpwssds
m512 mm512 4fmadd ps ( m512 a, m512i b0, m512i b1, m512i b2, m512i b3, m128i * c) v4fmaddps
```

另一个神奇的网站

- ▶ <https://gcc.godbolt.org>, 这个网站可以方便的查看源代码和编译出的汇编。当然你本地gdb查看也成。



The screenshot displays the Compiler Explorer interface. On the left, the C++ source code is shown:

```
1 void test(int*s,int n,int t)
2 {
3     for(int i=0;i<n;i++)s[i]=t;
4 }
```

On the right, the assembly output for x86_64 gcc 4.9.2 is displayed, showing instructions such as:

```
56 addi $1, %r8d
57 cmpl $6, %r9d
58 leal 0(,%r8,8), %ebx
59 jbe .L7
60 movl %edx, -20(%rbp)
61 leaq (%rdi,%r11,4), %r11
62 xorl %eax, %eax
63 vbroadcastss -20(%rbp), %ymm1
64 xorl %r9d, %r9d
65 .L8:
66 vpor (%r11,%rax), %ymm1, %ymm0
67 addi $1, %r9d
68 vmovdqa %ymm0, (%r11,%rax)
69 addq $32, %rax
70 cmpl %r9d, %r8d
71 ja .L8
72 addl %ebx, %ecx
73 cmpl %ebx, %r18d
74 je .L24
75 vzeroupper
76 .L7:
77 movslq %ecx, %rax
78 orl %edx, (%rdi,%rax,4)
79 leal 1(%rcx), %eax
80 cmpl %eax, %esi
81 jle .L25
82 cltq
83 nel %edx, (%rdi,%rax,4)
```

感谢大家的倾听

- ▶ 感谢CCF，广州市第二中学提供这个交流的平台
- ▶ 感谢辛勤的张鸽鸽和松松松提供的宝贵建议
- ▶ 感谢神奇橡树无私的教诲。膜神树者处处阿克，只因神树大人在他背后。不膜神树者违背了神树大人的旨意，神树大人必将降下天谴。
- ▶ 希望新的知识能够让大家的卡常水平越来越高！（捂脸）