

C++中虚函数(virtual function)到底有多慢

虚函数为什么慢，cpu 分支预测技术，虚函数到底要调用哪些汇编，虚函数实现的简单图示，虚函数不能内联，

印象中经常看到有人批评 C++的虚函数很慢，但是虚函数为什么慢，虚函数到底有多慢呢？

一、理论分析

虚函数慢的原因主要有三个：

1. 多了几条汇编指令（运行时得到对应类的函数的地址）
2. 影响 cpu 流水线
3. 编译器不能内联优化（仅在用父类引用或者指针调用时，不能内联）

先简单说下虚函数的实现，以下面测试代码中的 VirtualVector 类为例，VirtualVector 类的内存布局如下：

当在用父类的引用或者指针调用虚函数时，会先从该对象的头部取到虚函数的地址（C++ 标准规定虚函数表地址必须放最前），再从虚函数表中取到实际要调用的函数的地址，最终调用该函数，汇编代码大概如下：

[cpp] view plain copy

```
1.          sum += v.at(i);          //要调用 at 函数
2. 00CF1305 mov          eax,dword ptr [ebx]      //取到对象的虚函数表地址
3. 00CF1307 mov          edx,dword ptr [eax+4]    //取到实际 VirtualVector 类的 at
          函数地址，因为 at 是第二个虚函数，所以要+4，如果是 clear 则+8，push_back 则不加
4. 00CF130A push        esi                    //参数压栈
5. 00CF130B mov          ecx,ebx
6. 00CF130D call         edx                    //调用真正的 VirtualVector 类的 at
          函数
```

所以，我们可以看到调用虚函数，相比普通函数，实际上多了三条汇编指令（取虚表，取函数地址，call 调用）。

至于虚函数如何影响 cpu 的流水线，则主要是因为 call 指令，具体可以看这个网址的演示：

CPU 流水线的演示：http://www.pictutorials.com/Instruction_Pipeline.html

第 3 点，编译器不能内联优化，则是因为在得到子类的引用或者指针之前，根本不知道要调用哪一个函数，所以无从内联。

但是，要注意的是，对于子类直接调用虚函数，是可以内联优化的。如以下的代码，编译器是完全可以内联展开的。

[cpp] view plain copy

```
1. VirtualVector v(100);
2. v.push_back(1);
3. v.at(0);
```

二、实际测试

光说不练显然不行，下面用代码来测试下虚函数到底有多慢。

下面的代码只是测试用，不考虑细节。

Vector 类包装了一个数组，提供 **push_back**, **at**, **clear** 函数。

VirtualVector 类继承了 **IVector**，同样实现了 **push_back**, **at**, **clear** 函数，但是都是虚函数。

[cpp] view plain copy

```
1. #include <iostream>
2. #include <time.h>
3. #include <vector>
4. using namespace std;
5.
6. const int size = 100000000;
7.
8. class Vector{
9. private:
10.     int *array;
11.     int pos;
12. public:
13.     Vector(int size):array(new int[size]),pos(0)
14.     {
15.     }
16.     void push_back(int val)
17.     {
18.         array[pos++] = val;
19.     }
20.     int at(int i)
21.     {
22.         return array[i];
23.     }
24.     void clear()
25.     {
26.         pos = 0;
27.     }
28. };
29. class IVector{
```

```

30. public:
31.     virtual void push_back(int val) = 0;
32.     virtual int at(int i) = 0;
33.     virtual void clear() = 0;
34.     virtual ~IVector() {};
35. };
36.
37. class VirtualVector : public IVector{
38. public:
39.     int *array;
40.     int pos;
41. public:
42.     VirtualVector(int size):array(new int[size]),pos(0)
43.     {
44.     }
45.     void push_back(int val)
46.     {
47.         array[pos++] = val;
48.     }
49.     int at(int i)
50.     {
51.         return array[i];
52.     }
53.     void clear()
54.     {
55.         pos = 0;
56.     }
57.     ~VirtualVector()
58.     {
59.         if(array != NULL)
60.             delete array;
61.     }
62. };
63.
64. void testVectorPush(Vector& v){
65.     v.clear();
66.
67.     clock_t nTimeStart;    //计时开始
68.     clock_t nTimeStop;     //计时结束
69.     nTimeStart = clock();  //
70.     for(int i = 0; i < size; ++i)
71.     {
72.         v.push_back(i);
73.         //cout<<v.size()<<endl;

```

```
74.     }
75.     nTimeStop = clock();    //
76.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
    "<< endl;
77. }
78.
79. void testVectorAt(Vector& v)
80. {
81.     clock_t nTimeStart;      //计时开始
82.     clock_t nTimeStop;       //计时结束
83.     int sum = 0;
84.     nTimeStart = clock();    //
85.     for(int j = 0; j < 1; ++j)
86.     {
87.         for(int i = 0; i < size; ++i)
88.         {
89.             sum += v.at(i);
90.         }
91.     }
92.
93.     nTimeStop = clock();    //
94.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
    "<< endl;
95.     cout<<"sum:"<<sum<<endl;
96. }
97.
98. void testVirtualVectorPush(IVector& v)
99. {
100.     v.clear();
101.
102.     clock_t nTimeStart;      //计时开始
103.     clock_t nTimeStop;       //计时结束
104.     nTimeStart = clock();    //
105.     for(int i = 0; i < size; ++i)
106.     {
107.         v.push_back(i);
108.         //cout<<v.size()<<endl;
109.     }
110.     nTimeStop = clock();    //
111.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
    "<< endl;
112. }
113.
114. void testVirtualVectorAt(IVector& v)
```

```

115. {
116.     clock_t nTimeStart;    //计时开始
117.     clock_t nTimeStop;    //计时结束
118.     int sum = 0;
119.     nTimeStart = clock();  //
120.     for(int j = 0; j < 1; ++j)
121.     {
122.         for(int i = 0; i < size; ++i)
123.         {
124.             sum += v.at(i);
125.         }
126.     }
127.
128.     nTimeStop = clock();    //
129.     cout << "耗时: " << (double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC << "秒"
        << endl;
130.     cout << "sum: " << sum << endl;
131. }
132.
133. int main()
134. {
135.     cout << sizeof(VirtualVector) << endl;
136.
137.
138.     Vector *v = new Vector(size);
139.     VirtualVector *V = new VirtualVector(size);
140.
141.     cout << "testVectorPush:" << endl;
142.     testVectorPush(*v);
143.     testVectorPush(*v);
144.     testVectorPush(*v);
145.     testVectorPush(*v);
146.
147.     cout << "testVirtualVectorPush:" << endl;
148.     testVirtualVectorPush(*V);
149.     testVirtualVectorPush(*V);
150.     testVirtualVectorPush(*V);
151.     testVirtualVectorPush(*V);
152.
153.     cout << "testVectorAt:" << endl;
154.     testVectorAt(*v);
155.     testVectorAt(*v);
156.     testVectorAt(*v);
157.     testVectorAt(*v);

```

```

158.
159.     cout<<"testVirtualVectorAt:"<<endl;
160.     testVirtualVectorAt(*V);
161.     testVirtualVectorAt(*V);
162.     testVirtualVectorAt(*V);
163.     testVirtualVectorAt(*V);
164.
165.     return 0;
166. }

```

上面的是只有一层继承的情况时的结果，尽管从虚函数的实现角度来看，多层继承和一层继承调用虚函数的效率都是一样的。

但是为了测试结果更加可信，下面是一个 6 层继承的测试代码（为了防止编译器的优化，有很多垃圾代码）：

[\[cpp\] view plain copy](#)

```

1.  #include <iostream>
2.  #include <time.h>
3.  #include <vector>
4.  using namespace std;
5.
6.  const int size = 100000000;
7.
8.  class Vector{
9.  private:
10.     int *array;
11.     int pos;
12. public:
13.     Vector(int size):array(new int[size]),pos(0)
14.     {
15.     }
16.     void push_back(int val)
17.     {
18.         array[pos++] = val;
19.     }
20.     int at(int i)
21.     {
22.         return array[i];
23.     }
24.     void clear()
25.     {
26.         pos = 0;

```

```

27.     }
28. };
29. class IVector{
30. public:
31.     virtual void push_back(int val) = 0;
32.     virtual int at(int i) = 0;
33.     virtual void clear() = 0;
34.     virtual ~IVector() {};
35. };
36.
37. class VirtualVector1 : public IVector{
38. public:
39.     int *array;
40.     int pos;
41. public:
42.     VirtualVector1(int size):array(new int[size]),pos(0)
43.     {
44.     }
45.     void push_back(int val)
46.     {
47.         array[1] = val;
48.     }
49.     int at(int i)
50.     {
51.         return array[1];
52.     }
53.     void clear()
54.     {
55.         pos = 0;
56.     }
57.     ~VirtualVector1()
58.     {
59.         if(array != NULL)
60.             delete array;
61.     }
62. };
63.
64. class VirtualVector2 : public VirtualVector1{
65. public:
66.     VirtualVector2(int size):VirtualVector1(size)
67.     {
68.     }
69.     void push_back(int val)
70.     {

```

```
71.         array[2] = val;
72.     }
73.     int at(int i)
74.     {
75.         return array[2];
76.     }
77.     void clear()
78.     {
79.         pos = 0;
80.     }
81. };
82.
83. class VirtualVector3 : public VirtualVector2{
84. public:
85.     VirtualVector3(int size):VirtualVector2(size)
86.     {
87.     }
88.     void push_back(int val)
89.     {
90.         array[3] = val;
91.     }
92.     int at(int i)
93.     {
94.         return array[3];
95.     }
96.     void clear()
97.     {
98.         pos = 0;
99.     }
100. };
101.
102. class VirtualVector4 : public VirtualVector3{
103. public:
104.     VirtualVector4(int size):VirtualVector3(size)
105.     {
106.     }
107.     void push_back(int val)
108.     {
109.         array[4] = val;
110.     }
111.     int at(int i)
112.     {
113.         return array[4];
114.     }
```



```
115.     void clear()
116.     {
117.         pos = 0;
118.     }
119. };
120.
121. class VirtualVector5 : public VirtualVector4{
122. public:
123.     VirtualVector5(int size):VirtualVector4(size)
124.     {
125.     }
126.     void push_back(int val)
127.     {
128.         array[5] = val;
129.     }
130.     int at(int i)
131.     {
132.         return array[5];
133.     }
134.     void clear()
135.     {
136.         pos = 0;
137.     }
138. };
139.
140. class VirtualVector6 : public VirtualVector5{
141. public:
142.     VirtualVector6(int size):VirtualVector5(size)
143.     {
144.     }
145.     void push_back(int val)
146.     {
147.         array[6] = val;
148.     }
149.     int at(int i)
150.     {
151.         return array[6];
152.     }
153.     void clear()
154.     {
155.         pos = 0;
156.     }
157. };
158.
```

```

159. class VirtualVector : public VirtualVector6{
160. public:
161.     VirtualVector(int size):VirtualVector6(size)
162.     {
163.     }
164.     void push_back(int val)
165.     {
166.         array[pos++] = val;
167.     }
168.     int at(int i)
169.     {
170.         return array[i];
171.     }
172.     void clear()
173.     {
174.         pos = 0;
175.     }
176. };
177.
178. void testVectorPush(Vector& v){
179.     v.clear();
180.
181.     clock_t nTimeStart;    //计时开始
182.     clock_t nTimeStop;    //计时结束
183.     nTimeStart = clock();  //
184.     for(int i = 0; i < size; ++i)
185.     {
186.         v.push_back(i);
187.         //cout<<v.size()<<endl;
188.     }
189.     nTimeStop = clock();  //
190.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒
        "<< endl;
191. }
192.
193. void testVectorAt(Vector& v)
194. {
195.     clock_t nTimeStart;    //计时开始
196.     clock_t nTimeStop;    //计时结束
197.     int sum = 0;
198.     nTimeStart = clock();  //
199.     for(int j = 0; j < 1; ++j)
200.     {
201.         for(int i = 0; i < size; ++i)

```

```

202.     {
203.         sum += v.at(i);
204.     }
205. }
206.
207.     nTimeStop = clock();    //
208.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒
    "<< endl;
209.     cout<<"sum:"<<sum<<endl;
210. }
211.
212. void testVirtualVectorPush(IVector& v)
213. {
214.     v.clear();
215.     clock_t nTimeStart;      //计时开始
216.     clock_t nTimeStop;       //计时结束
217.     nTimeStart = clock();    //
218.     for(int i = 0; i < size; ++i)
219.     {
220.         v.push_back(i);
221.         //cout<<v.size()<<endl;
222.     }
223.     nTimeStop = clock();    //
224.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒
    "<< endl;
225. }
226.
227. void testVirtualVectorAt(IVector& v)
228. {
229.     clock_t nTimeStart;      //计时开始
230.     clock_t nTimeStop;       //计时结束
231.     int sum = 0;
232.     nTimeStart = clock();    //
233.     for(int j = 0; j < 1; ++j)
234.     {
235.         for(int i = 0; i < size; ++i)
236.         {
237.             sum += v.at(i);
238.         }
239.     }
240.
241.     nTimeStop = clock();    //
242.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒
    "<< endl;

```

```
243.     cout<<"sum:"<<sum<<endl;
244. }
245.
246. int main()
247. {
248.     cout<<sizeof(VirtualVector)<<endl;
249.     {
250.         auto v = VirtualVector1(size);
251.         v.push_back(0);
252.         cout<<v.at(0)<<endl;
253.     }
254.     {
255.         auto v = VirtualVector2(size);
256.         v.push_back(0);
257.         cout<<v.at(0)<<endl;
258.     }
259.     {
260.         auto v = VirtualVector3(size);
261.         v.push_back(0);
262.         cout<<v.at(0)<<endl;
263.     }
264.     {
265.         auto v = VirtualVector4(size);
266.         v.push_back(0);
267.         cout<<v.at(0)<<endl;
268.     }
269.     {
270.         auto v = VirtualVector5(size);
271.         v.push_back(0);
272.         cout<<v.at(0)<<endl;
273.     }
274.     {
275.         auto v = VirtualVector6(size);
276.         v.push_back(0);
277.         cout<<v.at(0)<<endl;
278.     }
279.
280.     auto *v = new Vector(size);
281.     auto *V = new VirtualVector(size);
282.
283.     cout<<"testVectorPush:"<<endl;
284.     testVectorPush(*v);
285.     testVectorPush(*v);
286.     testVectorPush(*v);
```

```

287.     testVectorPush(*v);
288.
289.     cout<<"testVirtualVectorPush:"<<endl;
290.     testVirtualVectorPush(*V);
291.     testVirtualVectorPush(*V);
292.     testVirtualVectorPush(*V);
293.     testVirtualVectorPush(*V);
294.
295.     cout<<"testVectorAt:"<<endl;
296.     testVectorAt(*v);
297.     testVectorAt(*v);
298.     testVectorAt(*v);
299.     testVectorAt(*v);
300.
301.     cout<<"testVirtualVectorAt:"<<endl;
302.     testVirtualVectorAt(*V);
303.     testVirtualVectorAt(*V);
304.     testVirtualVectorAt(*V);
305.     testVirtualVectorAt(*V);
306.
307.     return 0;
308. }

```

测试结果：

测试结果都取最小时间

1 层继承的测试结果：

	push_back	at
Vector	0.263s	0.04s
VirtualVector	0.331s	0.222s
倍数	1.25	5.55

6 层继承的测试结果：

	push_back	at
Vector	0.262s	0.041s
VirtualVector	0.334s	0.223s
倍数	1.27	5.43

一、可以看出继承层数和虚函数调用效率无关

二、可以看出虚函数慢得有点令人发指了，对于 **at** 操作，虚函数花的时间竟然是普通函数的 5.5 倍！

但是，再看看，我们可以发现对于 `push_back` 操作，虚函数花的时间是普通函数的 1.25 倍。

why?

再分析下代码，我们可以发现 `at` 操作的逻辑，明显要比 `push_back` 的逻辑要简单。

虚函数额外消耗时间为 `vt`，函数本身所消耗时间为 `ft`，则有以下

倍数 = $(vt + ft)/ft = 1 + vt/ft$

显然当 `ft` 越大，即函数本身消耗时间越长，则倍数越小。

那么让我们在 `at` 操作中加了额外代码，统计下 1 到 100 之和：

[\[cpp\] view plain copy](#)

```
1. int at(int i)
2. {
3.     sssForTest = 0;
4.     for(int j = 0; j < 100; ++j)
5.         sssForTest += j;
6.     return array[i];
7. }
```

测试代码：

[\[cpp\] view plain copy](#)

```
1. #include <iostream>
2. #include <time.h>
3. #include <vector>
4. using namespace std;
5.
6. const int size = 100000000;
7. int sssForTest = 0;
8.
9. class Vector{
10. private:
11.     int *array;
12.     int pos;
13. public:
14.     Vector(int size):array(new int[size]),pos(0)
15.     {
16.     }
17.     void push_back(int val)
18.     {
19.         array[pos++] = val;
20.     }
21.     int at(int i)
```

```

22.     {
23.         sssForTest = 0;
24.         for(int j = 0; j < 100; ++j)
25.             sssForTest += j;
26.         return array[i];
27.     }
28.     void clear()
29.     {
30.         pos = 0;
31.     }
32. };
33. class IVector{
34. public:
35.     virtual void push_back(int val) = 0;
36.     virtual int at(int i) = 0;
37.     virtual void clear() = 0;
38.     virtual ~IVector() {};
39. };
40.
41. class VirtualVector : public IVector{
42. public:
43.     int *array;
44.     int pos;
45. public:
46.     VirtualVector(int size):array(new int[size]),pos(0)
47.     {
48.     }
49.     void push_back(int val)
50.     {
51.         array[pos++] = val;
52.     }
53.     int at(int i)
54.     {
55.         sssForTest = 0;
56.         for(int j = 0; j < 100; ++j)
57.             sssForTest += j;
58.         return array[i];
59.     }
60.     void clear()
61.     {
62.         pos = 0;
63.     }
64.     ~VirtualVector()
65.     {

```

```
66.         if(array != NULL)
67.             delete array;
68.     }
69. };
70.
71. void testVectorPush(Vector& v){
72.     v.clear();
73.
74.     clock_t nTimeStart;        //计时开始
75.     clock_t nTimeStop;         //计时结束
76.     nTimeStart = clock();      //
77.     for(int i = 0; i < size; ++i)
78.     {
79.         v.push_back(i);
80.         //cout<<v.size()<<endl;
81.     }
82.     nTimeStop = clock();       //
83.     cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
        "<< endl;
84. }
85.
86. void testVectorAt(Vector& v)
87. {
88.     clock_t nTimeStart;        //计时开始
89.     clock_t nTimeStop;         //计时结束
90.     int sum = 0;
91.     nTimeStart = clock();      //
92.     for(int j = 0; j < 1; ++j)
93.     {
94.         for(int i = 0; i < size; ++i)
95.         {
96.             sum += v.at(i);
97.         }
98.     }
99.
100.    nTimeStop = clock();       //
101.    cout <<"耗时: "<<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
        "<< endl;
102.    cout<<"sum:"<<sum<<endl;
103. }
104.
105. void testVirtualVectorPush(IVector& v)
106. {
107.     v.clear();
```



```
108.
109.     clock_t nTimeStart;        //计时开始
110.     clock_t nTimeStop;         //计时结束
111.     nTimeStart = clock();      //
112.     for(int i = 0; i < size; ++i)
113.     {
114.         v.push_back(i);
115.         //cout<<v.size()<<endl;
116.     }
117.     nTimeStop = clock();       //
118.     cout <<"耗时: " <<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
        "<< endl;
119. }
120.
121. void testVirtualVectorAt(IVector& v)
122. {
123.     clock_t nTimeStart;        //计时开始
124.     clock_t nTimeStop;         //计时结束
125.     int sum = 0;
126.     nTimeStart = clock();      //
127.     for(int j = 0; j < 1; ++j)
128.     {
129.         for(int i = 0; i < size; ++i)
130.         {
131.             sum += v.at(i);
132.         }
133.     }
134.
135.     nTimeStop = clock();       //
136.     cout <<"耗时: " <<(double)(nTimeStop - nTimeStart)/CLOCKS_PER_SEC<<"秒"
        "<< endl;
137.     cout<<"sum:"<<sum<<endl;
138. }
139.
140. int main()
141. {
142.     cout<<sizeof(VirtualVector)<<endl;
143.
144.
145.     Vector *v = new Vector(size);
146.     VirtualVector *V = new VirtualVector(size);
147.
148.     cout<<"testVectorPush:"<<endl;
149.     testVectorPush(*v);
```

```

150.    testVectorPush(*v);
151.    testVectorPush(*v);
152.    testVectorPush(*v);
153.
154.    cout<<"testVirtualVectorPush:"<<endl;
155.    testVirtualVectorPush(*V);
156.    testVirtualVectorPush(*V);
157.    testVirtualVectorPush(*V);
158.    testVirtualVectorPush(*V);
159.
160.    cout<<"testVectorAt:"<<endl;
161.    testVectorAt(*v);
162.    testVectorAt(*v);
163.    testVectorAt(*v);
164.    testVectorAt(*v);
165.
166.    cout<<"testVirtualVectorAt:"<<endl;
167.    testVirtualVectorAt(*V);
168.    testVirtualVectorAt(*V);
169.    testVirtualVectorAt(*V);
170.    testVirtualVectorAt(*V);
171.
172.    return 0;
173. }

```

at 操作中增加求和后的统计结果:

	push_back	at 增 加求和 代码
Vector	0.265s	6.893s
VirtualVector	0.328s	7.125s
倍数	1.23	1.03

只是简单增加了一个求和代码，我们可以看到，倍数变成了 1.03，也就是说虚函数的消耗基本可以忽略了。

所以说，虚函数的效率到底低不低和实际要调用的函数的耗时有关，当函数本身的的耗时越长，则虚函数的影响则越小。

再从另一个角度来看，一个虚函数调用到底额外消耗了多长时间？

从统计数据来看 100,000,000 次函数调用，虚函数总共额外消耗了 0.05~0.23 秒

（VirtualVector 对应时间减去 Vector 时间），

也就是说，1 亿次调用，虚函数额外花的时间是 0.x 到 2.3 秒。

也就是说，如果你有个函数，要被调用 1 亿次，而这 1 亿次调用所花的时间是几秒，十几秒，且你不能容忍它慢一二秒，那么就干掉虚函数吧^_^。

三、总结：

1. 虚函数调用效率和继承层数无关；
2. 其实虚函数还是挺快的。
3. 如果真的要完全移除虚函数，那么如果要实现运行时多态，则要用到函数指针，据上面的分析，函数指针基本具有虚函数的所有缺点（要传递函数指针，同样无法内联，同样影响流水线），且函数指针会使代码混乱。

BTW：测试 cpu 是 i5。

TODO：测试指针函数，boost::bind 的效率