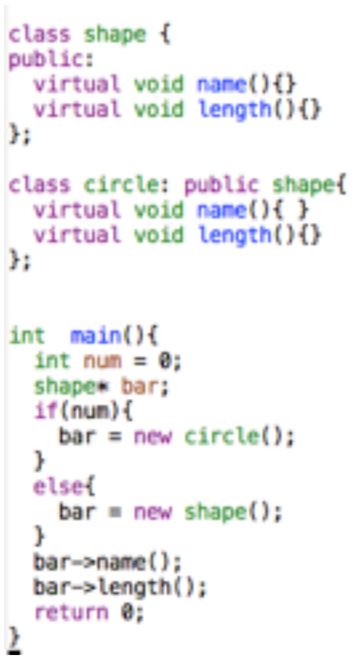
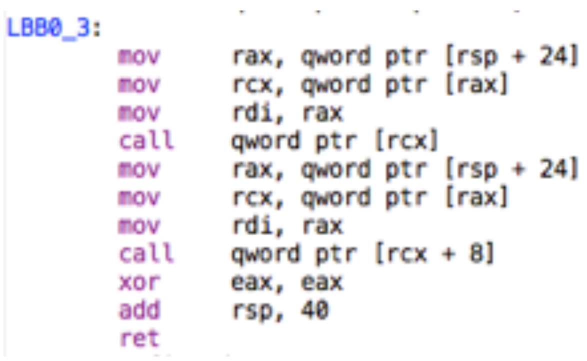
Pat Wongwiset
nw9ca
11/16/17
postlab09.pdf

Dynamic Dispatch



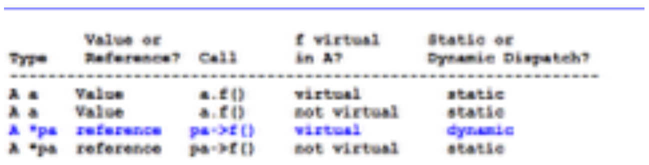
```
class shape {
public:
  virtual void name(){}
  virtual void length(){}
};

class circle: public shape{
  virtual void name(){ }
  virtual void length(){}
};


int  main(){
  int num = 0;
  shape* bar;
  if(num){
    bar = new circle();
  }
  else{
    bar = new shape();
  }
  bar->name();
  bar->length();
  return 0;
}
```



```
LBB0_3:
        mov     rax, qword ptr [rsp + 24]
        mov     rcx, qword ptr [rax]
        mov     rdi, rax
        call    qword ptr [rcx]
        mov     rax, qword ptr [rsp + 24]
        mov     rcx, qword ptr [rax]
        mov     rdi, rax
        call    qword ptr [rcx + 8]
        xor     eax, eax
        add     rsp, 40
        ret
```

The virtual method is powerful when the program cannot determine which method it should use until runtime. The program would pass the virtual method by calling from the address, not from the method's name directly. There exists a virtual table which stores all the virtual methods in the program and can be pointed by a register, rcx from the code above. The virtual method in the virtual table is offset by 8 bytes in the virtual table. Since all virtual methods are stored in the virtual table as a memory address, a subclass can change the behavior of the base class directly.

When does C++ use Dynamic Dispatch



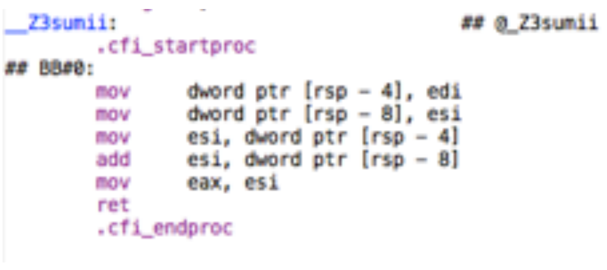
Type	Value or Reference?	Call	f virtual in A?	Static or Dynamic Dispatch?
A a	Value	a.f()	virtual	static
A a	Value	a.f()	not virtual	static
A *pa	reference	pa->f()	virtual	dynamic
A *pa	reference	pa->f()	not virtual	static

http://condor.depaul.edu/ichu/csc447/notes/wk10/Dynamic2.htm

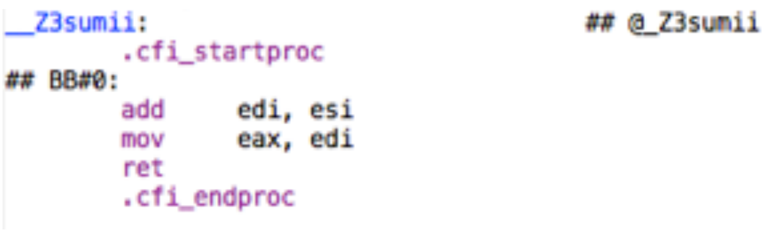
The program will use dynamic dispatch when the method can be passed by memory address.

<u>Optimized code</u>

Passing parameter into a function



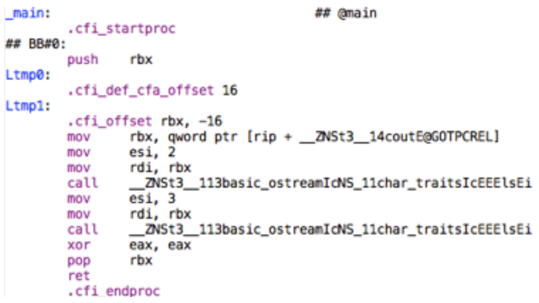
```
__Z3sumii:                              ## @__Z3sumii
        .cfi_startproc
## BB#0:
        mov     dword ptr [rsp - 4], edi
        mov     dword ptr [rsp - 8], esi
        mov     esi, dword ptr [rsp - 4]
        add     esi, dword ptr [rsp - 8]
        mov     eax, esi
        ret
        .cfi_endproc
```



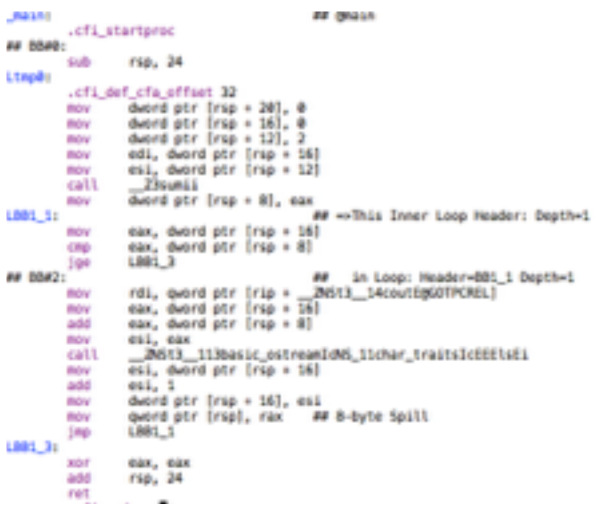
```
__Z3sumii:                              ## @__Z3sumii
        .cfi_startproc
## BB#0:
        add     edi, esi
        mov     eax, edi
        ret
        .cfi_endproc
```

The optimized code does not allocate local variable if unnecessary, as compared above. Normal code uses [rsp - 4] and [rsp -8] to operate the function and then move final result to return at the end, but the optimized code adjust value directly in register.

Loop



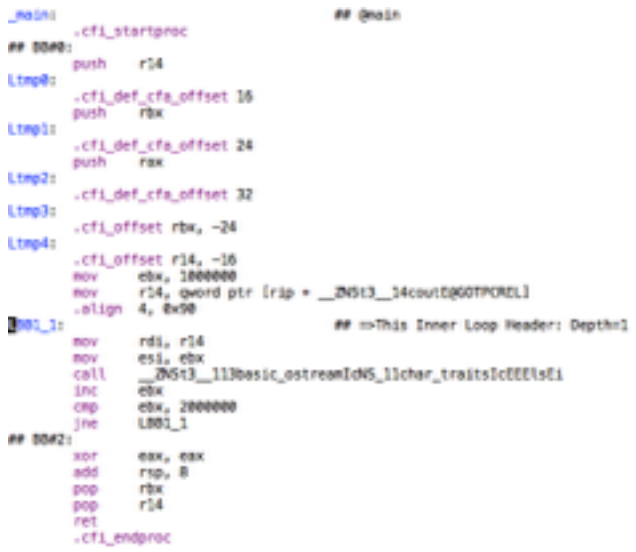
```
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        push    rbx
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset rbx, -16
        mov     rbx, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
        mov     esi, 2
        mov     rdi, rbx
        call    __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
        mov     esi, 3
        mov     rdi, rbx
        call    __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
        xor     eax, eax
        pop     rbx
        ret
        .cfi_endproc
```



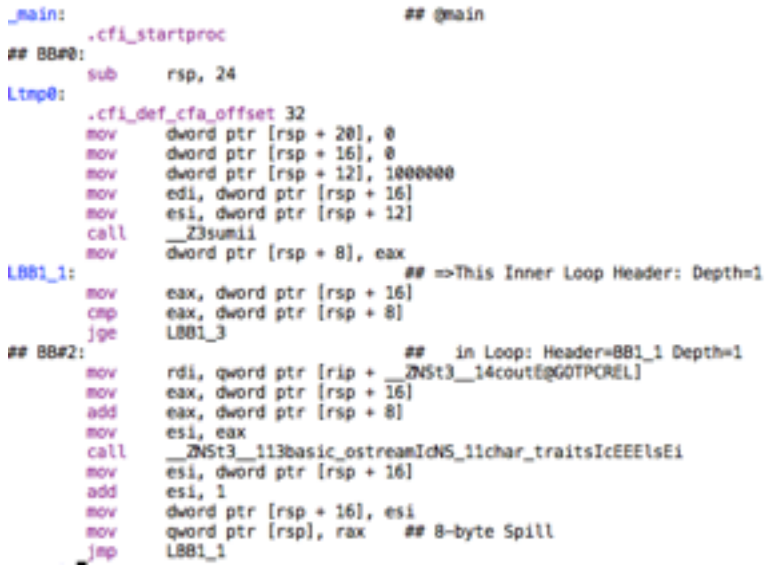
```
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        sub     rsp, 24
Ltmp0:
        .cfi_def_cfa_offset 32
        mov     dword ptr [rsp + 20], 0
        mov     dword ptr [rsp + 16], 0
        mov     dword ptr [rsp + 12], 2
        mov     edi, dword ptr [rsp + 16]
        mov     esi, dword ptr [rsp + 12]
        call    __Z3sumii
        mov     dword ptr [rsp + 8], eax
LBB1_1:                                 ## =>This Inner Loop Header: Depth=1
        mov     eax, dword ptr [rsp + 16]
        cmp     eax, dword ptr [rsp + 8]
        jge     LBB1_3
## BB#2:                                ##   in Loop: Header=BB1_1 Depth=1
        mov     rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
        mov     eax, dword ptr [rsp + 16]
        add     eax, dword ptr [rsp + 8]
        mov     esi, eax
        call    __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
        mov     esi, dword ptr [rsp + 16]
        add     esi, 1
        mov     dword ptr [rsp + 16], esi
        mov     qword ptr [rsp], rax    ## 8-byte Spill
        jmp     LBB1_1
LBB1_3:
        xor     eax, eax
        add     rsp, 24
        ret
```

When the iterating number for loop is small (from the code above n = 2), optimized code will iterate by implementing code inside the loop directly, not looping.

Also, the optimized code will try to not use the number of local variables. It will maximize the number of registers i.e. it uses rbx instead of using stack to allocate local variables in the code above.



```
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        push    r14
Ltmp0:
        .cfi_def_cfa_offset 16
        push    rbx
Ltmp1:
        .cfi_def_cfa_offset 24
        push    rax
Ltmp2:
        .cfi_def_cfa_offset 32
Ltmp3:
        .cfi_offset rbx, -24
Ltmp4:
        .cfi_offset r14, -16
        mov     ebx, 1000000
        mov     r14, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
        .align  4, 0x90
LBB1_1:                                 ## =>This Inner Loop Header: Depth=1
        mov     rdi, r14
        mov     esi, ebx
        call    __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
        inc     ebx
        cmp     ebx, 2000000
        jne     LBB1_1
## BB#2:
        xor     eax, eax
        add     rsp, 8
        pop     rbx
        pop     r14
        ret
        .cfi_endproc
```



```
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        sub     rsp, 24
Ltmp0:
        .cfi_def_cfa_offset 32
        mov     dword ptr [rsp + 20], 0
        mov     dword ptr [rsp + 16], 0
        mov     dword ptr [rsp + 12], 1000000
        mov     edi, dword ptr [rsp + 16]
        mov     esi, dword ptr [rsp + 12]
        call    __Z3sumii
        mov     dword ptr [rsp + 8], eax
LBB1_1:                                 ## =>This Inner Loop Header: Depth=1
        mov     eax, dword ptr [rsp + 16]
        cmp     eax, dword ptr [rsp + 8]
        jge     LBB1_3
## BB#2:                                ##   in Loop: Header=BB1_1 Depth=1
        mov     rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
        mov     eax, dword ptr [rsp + 16]
        add     eax, dword ptr [rsp + 8]
        mov     esi, eax
        call    __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
        mov     esi, dword ptr [rsp + 16]
        add     esi, 1
        mov     dword ptr [rsp + 16], esi
        mov     qword ptr [rsp], rax    ## 8-byte Spill
        jmp     LBB1_1
```

When n (iterating number) becomes large, the optimized code will move/copy the local variable (rip + _Znst3_14coutE@GOTPCREL) to register r14 before going to the loop so that the program will run faster due to reducing the number of "dereference".

There exists only one jump (jne) in the optimized code while the normal code have jge and jmp. Thus, the program can run faster because it does not have to jump back to check the loop condition again.

(https://books.google.com/books?id=LZ7VAwAAQBAJ&printsec=frontcover#v=onepage&q&f=false :
Visual C++ Optimization with Assembly Code)