Ashley Nguyen
PostLab09 - postlab9.pdf
4/13/16

Dynamic Dispatch

In object oriented programming, when setting up fields of the superclass to be "inherited" by the subclass, dynamic dispatch is implemented when the keyword *virtual* is used. Dynamic dispatch is most useful when two functions/attributes have the exact same name in the superclass and the subclass. Dynamic dispatch makes it clearer which function call from which class will be used at run-time. If the keyword virtual is preceded by the members of a certain field (i.e. public, private, protected) then the compiler will check to see if the subclass redefines this method at run time. If the subclass does indeed redefine this method then the subclass's methods will be the ones that are used to run the program. Dynamic dispatch is the process in dynamic binding, which operates at runtime to search for the proper function and "bind" to it. However, some trade-offs occur when deciding to use dynamic dispatch or static dispatch. Although dynamic dispatch is flexible and lends to optimization, it can be costly in terms of time. This is because dynamic dispatch requires a lookup time to find the function and a checking time to see if the subclass has redefined the function (attribute accessing). To investigate this topic further, I created a Person class as the superclass and the id class is the subclass. These two classes have two methods that are identical, the setName method takes in a string parameter and the foo() method. The main method shows the ambiguity in which setName() and which foo() method the id class will inherit. Looking at the assembly may provide answers.

```
//Superclass
class Person {
public:
  Person(void) : name(""){}
  ~Person(void){}
  virtual void setName( string n ){
    name = n;
  }
  virtual void foo(){
    bar = 100;
    cout << bar << endl;
  }
  void print(void){
    cout << name << endl;
  }

private:
  string name;
  int bar;
};
```

```
//Subclass - class id treats Person's name
class id: public Person {
public:
  virtual void setName( string n ){
    numberID = n;
  }
  virtual void foo(){
    bar = 1;
    cout << bar << endl;
  }
  void print(void){
    cout << numberID << endl;
  }


private:
  string numberID;
  int bar;
};
```

```
//main
  int main(){
    id ash;
    ash.setName("Ashley");
    ash.print();

    ash.foo();
    return 0;
  }
```

```
_ZN6Person7setNameESs:
.LFB981:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 24
        mov     eax, DWORD PTR [ebp+8]
        lea     edx, [eax+4]
        mov     eax, DWORD PTR [ebp+12]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], edx
        call    _ZNSsaSERKSs
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

```
_ZN2id7setNameESs:
.LFB984:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 24
        mov     eax, DWORD PTR [ebp+8]
        lea     edx, [eax+12]
        mov     eax, DWORD PTR [ebp+12]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], edx
        call    _ZNSsaSERKSs
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

```
main:
.LFB987:
        push    ebp
        mov     ebp, esp
        push    ebx
        and     esp, -16
        sub     esp, 48
        lea     eax, [esp+28]
        mov     DWORD PTR [esp], eax
.LEHB8:
        call    _ZN2idC1Ev
.LEHE8:
        lea     eax, [esp+23]
        mov     DWORD PTR [esp], eax
        call    _ZNSaIcEC1Ev
        lea     eax, [esp+23]
        mov     DWORD PTR [esp+8], eax
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC1
        lea     eax, [esp+24]
        mov     DWORD PTR [esp], eax
.LEHB9:
        call    _ZNSsC1EPKcRKSaIcE
```

As shown in the Person class and id class assembly of the setName() method, the assembly code is similar at the machine level. However, at runtime in the main method, under .LEHB8 header (see arrow), the id class is called when calling the setName() method. This shows that using the virtual keyword will enable dynamic dispatch so that the compiler knows to use the subclass's redefined method instead of the superclass's method. Thus in the Virtual Method Table, the value stored in that portion of the stack will reference the memory address of the subclass rather than the superclass. Similar findings would also be shown for the identical foo() method as well. It is important to note that subclasses can redefine functions in contrast to the superclass and they can redefine attributes as well. Thus in term of the VMT, a function pointer normally is located before the lookup table for the superclass. But when the subclass function is called instead, the pointer jumps a predefined offset (from the superclass) and is positioned before the lookup table of the function that will actually be called (id class in my example),. Furthermore, in the case where the similarly constructed methods perform completely different functions, dynamic dispatch has the ability to exclusively access instances of the altered code in the subclass. This may be helpful in compacting and aligning which bumps up the speed on RISC architectures (Milton, Schmidt). However, the cost of size and compilation time is a result of altering an attribute from the parent class. Whereas, static dispatch (strictly accessing attributes of parent class) is hardwired into the compiler in which it would not have to take any extra measures (lookup/checking) to access the intended function.

Optimized Code

```
int calculator(char sign, int a, int b){
  int result = 0;
  if( sign == '+' ){
    result = a + b;
  }
  else if(sign == '-' ){
    result = a - b;
  }
  else if( sign == '*' ){
    result = a * b;
  }
  else if( sign == '/' ){
    result = a/b;
  }
  else if( sign == '^' ){
    result = 1;
    for( int i = 0; i < b; i++){
      result = a * result;
    }
  }
  return result;
}
```

```
int main(){
  char function;
  int x;
  int y;
  cout << "Enter an operation: " << endl;
  cin >> function;

  cout << "Enter first int: " << endl;
  cin >> x;

  cout << "Enter second int: " << endl;
  cin >> y;

  cout << "Peforming operation..." << endl;
  cout << "Your result is: " << calculator( function, x, y ) << endl;
  return 0;
}
```

I created a calculator function that implemented multiple if statements to determine which operation to perform. In addition, to determine how loops are optimized I created a power operation that iteratively calculates the power. The main function takes in user input to perform a calculation on two integers.

Firstly, the optimized assembly for the calculator function demonstrates that the function can operate appropriately with less lines of code. This is because the -O2 flag condenses the code by getting rid of unnecessary lines and registers such as push ebp and mov ebp, esp. Instead in optimized assembly, the callee-saved registers are used directly to push the parameters through solely the stack pointer. This saves a couple of lines by not using ebp.

Another difference is the use of the least significant bits (i.e. AL, BL, CL, DL) rather than BYTE PTR when making the comparisons if whether the sign == ‘operator’. By using only a portion of the register to call cmp on, this optimization enables to compiler to search a specific part of the register which optimizes space and compile-time.

Furthermore, by utilizing the EBX register as an updated register of local variables (unoptimized DWORD PTR[ebp +20]), less comparisons need to be made from varying offsets from EBP which optimizes the number of registers that are actually used on the stack.

In addition, the optimized code makes use of the LEA command more often in the main method to effectively perform the operation (such as add) in one line by adding the memory addresses of the appropriate two registers together that hold parameters. By dealing with addresses themselves, loading the effective address reduces the need for multiple offset base pointers to continuously point to various positions on the stack to refer to a certain register.

Unoptimized assembly uses varying jump cases (jne and jmp) to jump to the appropriate if statement depending on the first parameter. However, optimized assembly for the most part uniformly uses a jump equals (je) to get to its proper if statement by deterministic comparisons to the least significant 8 bits as specified above. Instead of instantiating unneeded registers in order to make a comparison, the optimized code compares the values (specifically the least significant bits) directly with the registers in use to simplify the jump and reduce the need to mov, compare and jump to just compare and jump. This fairly consistent logic behind each jump case optimizes the assembly by making it clearer to the compiler where to jump for each case in a linear-like fashion.

Optimized assembly makes use of the xor command in place of the mov command. Rather than move a variable holding the value of 0 into a register to reset it, xor completes this in practically constant time. Exclusive-or ensures by boolean logic automatically resets the register to 0 which is faster than moving one value into a register.

In conclusion, optimized code renders assembly code more succinct by reformatting the way loops, if-statements and registers are handled. Construction and destruction calls (prologue and epilogue) are minimized to use relevant stack pointers and registers that can be reused throughout the function to hold varying values (such as callee-saved registers). For my code, since it was quite lengthy, function calls did not vary that much from unoptimized code besides the compacted use of registers to handle parameters. This exploration revealed a multitude of assembly intricacies that optimize compiler time and memory space in certain cases.

Unoptimized Assembly - 218 lines

```
_Z10calculatorcii:
.LFB975:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 20
        mov     eax, DWORD PTR [ebp+8]
        mov     BYTE PTR [ebp-20], al
        mov     DWORD PTR [ebp-8], 0
        cmp     BYTE PTR [ebp-20], 43
        jne     .L2
        mov     eax, DWORD PTR [ebp+16]
        mov     edx, DWORD PTR [ebp+12]
        add     eax, edx
        mov     DWORD PTR [ebp-8], eax
        jmp     .L3
.L2:
        cmp     BYTE PTR [ebp-20], 45
        jne     .L4
        mov     eax, DWORD PTR [ebp+16]
        mov     edx, DWORD PTR [ebp+12]
        sub     edx, eax
        mov     eax, edx
        mov     DWORD PTR [ebp-8], eax
        jmp     .L3
.L4:
        cmp     BYTE PTR [ebp-20], 42
        jne     .L5
        mov     eax, DWORD PTR [ebp+12]
        imul    eax, DWORD PTR [ebp+16]
        mov     DWORD PTR [ebp-8], eax
        jmp     .L3
.L5:
        cmp     BYTE PTR [ebp-20], 47
        jne     .L6
        mov     eax, DWORD PTR [ebp+12]
        cdq
        idiv    DWORD PTR [ebp+16]
        mov     DWORD PTR [ebp-8], eax
        jmp     .L3
.L6:
        cmp     BYTE PTR [ebp-20], 94
        jne     .L3
        mov     DWORD PTR [ebp-8], 1
        mov     DWORD PTR [ebp-4], 0
        jmp     .L7
.L8:
        mov     eax, DWORD PTR [ebp-8]
        imul    eax, DWORD PTR [ebp+12]
        mov     DWORD PTR [ebp-8], eax
        add     DWORD PTR [ebp-4], 1
.L7:
        mov     eax, DWORD PTR [ebp-4]
        cmp     eax, DWORD PTR [ebp+16]
        jl      .L8
.L3:
        mov     eax, DWORD PTR [ebp-8]
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

```
main:
.LFB976:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        push    ebx
        and     esp, -16
        sub     esp, 32
        .cfi_offset 3, -12
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC0
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        lea     eax, [esp+23]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
        call    _ZStrsIcSt11char_traitsIcEERSt13basic_istreamIT_T0_ES6_RS3_
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC1
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC1
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        lea     eax, [esp+24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
        call    _ZNSirsERi
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC2
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        lea     eax, [esp+28]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
        call    _ZNSirsERi
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC3
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        mov     ecx, DWORD PTR [esp+28]
        call    _ZNSolsEPFRSoS_E
        mov     ecx, DWORD PTR [esp+28]
        mov     edx, DWORD PTR [esp+24]
        movzx   eax, BYTE PTR [esp+23]
        movsx   eax, al
        mov     DWORD PTR [esp+8], ecx
        mov     DWORD PTR [esp+4], edx
        mov     DWORD PTR [esp], eax
        call    _Z10calculatorcii
        mov     ebx, eax
        mov     DWORD PTR [esp+4], OFFSET FLAT:.LC4
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
        mov     DWORD PTR [esp+4], ebx
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEi
        mov     DWORD PTR [esp+4], OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        mov     eax, 0
        mov     ebx, DWORD PTR [ebp-4]
        leave
        .cfi_restore 5
        .cfi_restore 3
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

Optimized Assembly -201 lines

```
_Z10calculatorcii:
.LFB1015:
	.cfi_startproc
	push	ebx
	.cfi_def_cfa_offset 8
	.cfi_offset 3, -8
	mov	ecx, DWORD PTR [esp+8]
	mov	edx, DWORD PTR [esp+12]
	mov	ebx, DWORD PTR [esp+16]
	cmp	cl, 43
	lea	eax, [edx+ebx]
	je	.L3
	mov	eax, edx
	sub	eax, ebx
	cmp	cl, 45
	je	.L3
	cmp	cl, 42
	je	.L12
	cmp	cl, 47
	je	.L13
	xor	eax, eax
	cmp	cl, 94
	je	.L14
.L3:
	pop	ebx
	.cfi_remember_state
	.cfi_restore 3
	.cfi_def_cfa_offset 4
	.p2align 4,,2
	ret
	.p2align 4,,7
	.p2align 3
.L14:
	.cfi_restore_state
	test	ebx, ebx
	jle	.L9
	xor	ecx, ecx
	mov	al, 1
	.p2align 4,,7
	.p2align 3
.L7:
	add	ecx, 1
	imul	eax, edx
	cmp	ecx, ebx
	jne	.L7
	pop	ebx
	.cfi_remember_state
	.cfi_restore 3
	.cfi_def_cfa_offset 4
	ret
	.p2align 4,,7
	.p2align 3
.L13:
	.cfi_restore_state
	mov	eax, edx
	cdq
	idiv	ebx
	pop	ebx
	.cfi_remember_state
	.cfi_restore 3
	.cfi_def_cfa_offset 4
	ret
	.p2align 4,,7
	.p2align 3
.L12:
	.cfi_restore_state
	mov	eax, edx
	imul	eax, ebx
	pop	ebx
	.cfi_remember_state
	.cfi_restore 3
	.cfi_def_cfa_offset 4
	ret
.L9:
	.cfi_restore_state
	mov	eax, 1
	pop	ebx
	.cfi_restore 3
	.cfi_def_cfa_offset 4
	ret
	.cfi_endproc
```

```
main:
.LFB1016:
	.cfi_startproc
	push	ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	mov	ebp, esp
	.cfi_def_cfa_register 5
	push	ebx
	and	esp, -16
	sub	esp, 32
	.cfi_offset 3, -12
	mov	DWORD PTR [esp+4], OFFSET FLAT:.LC0
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	call	_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
	mov	DWORD PTR [esp], eax
	call	_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
	lea	eax, [esp+23]
	mov	DWORD PTR [esp+4], eax
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
	call	_ZStrsIcSt11char_traitsIcEERSt13basic_istreamIT_T0_ES6_RS3_
	mov	DWORD PTR [esp+4], OFFSET FLAT:.LC1
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	call	_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
	mov	DWORD PTR [esp], eax
	call	_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
	lea	eax, [esp+24]
	mov	DWORD PTR [esp+4], eax
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
	call	_ZNSirsERi
	mov	DWORD PTR [esp+4], OFFSET FLAT:.LC2
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	call	_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
	mov	DWORD PTR [esp], eax
	call	_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
	lea	eax, [esp+28]
	mov	DWORD PTR [esp+4], eax
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt3cin
	call	_ZNSirsERi
	mov	DWORD PTR [esp+4], OFFSET FLAT:.LC3
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	call	_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
	mov	DWORD PTR [esp], eax
	call	_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
	mov	eax, DWORD PTR [esp+28]
	mov	DWORD PTR [esp+8], eax
	mov	eax, DWORD PTR [esp+24]
	mov	DWORD PTR [esp+4], eax
	movsx	eax, BYTE PTR [esp+23]
	mov	DWORD PTR [esp], eax
	call	_Z10calculatorcii
	call	_Z10calculatorcii
	mov	DWORD PTR [esp+4], OFFSET FLAT:.LC4
	mov	DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
	mov	ebx, eax
	call	_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
	mov	DWORD PTR [esp+4], ebx
	mov	DWORD PTR [esp], eax
	call	_ZNSolsEi
	mov	DWORD PTR [esp], eax
	call	_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
	xor	eax, eax
	mov	ebx, DWORD PTR [ebp-4]
	leave
	.cfi_restore 5
	.cfi_restore 3
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
```

Works Cited

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.5597&rep=rep1&type=pdf

http://www.agner.org/optimize/optimizing_assembly.pdf