Diren D Bharwani, 2002216, diren.dbharwani

Comparison of Assembly Code for Polynomial Computation Algorithms

opt_poly (32-bit)

```
poly_opt:
.LFB0:
  .cfi_startproc
  endbr32
  pushl %edi
  .cfi_def_cfa_offset 8
  .cfi_offset 7, -8
  pushl %esi
  .cfi_def_cfa_offset 12
  .cfi_offset 6, -12
  pushl %ebx
  .cfi_def_cfa_offset 16
  .cfi_offset 3, -16
  subl  $40, %esp
  .cfi_def_cfa_offset 56
  fldl  60(%esp)
  movl  68(%esp), %esi
  movl  56(%esp), %edx
  leal  -1(%esi), %eax
  fld %st(0)
  fmul  %st(1), %st
  fstpl (%esp)
  fldl  (%esp)
  fstpt 8(%esp)
  fldt  8(%esp)
  fmul  %st(1), %st
  fstpl (%esp)
  fldl  (%esp)
  fstpt 24(%esp)
  fldt  24(%esp)
  fmul  %st(1), %st
  fstpl (%esp)
  fldl  (%esp)
  fld %st(0)
  fmul  %st(2), %st
  fstpl (%esp)
  fldl  (%esp)
  fld %st(0)
  fmul  %st(3), %st
  fstpl (%esp)
  fldl  (%esp)
  fld %st(0)
  fmul  %st(4), %st
  fstpl (%esp)
  fldl  (%esp)
  fld %st(0)
  fmul  %st(5), %st
  fstpl (%esp)
```

```
  fldl  (%edx,%esi,8)
  cmpl  $6, %eax
  jle .L10
  fldl  (%esp)
  fxch  %st(1)
  leal  -8(%esi), %ebx
  leal  0(,%esi,8), %ecx
  shrl  $3, %ebx
  leal  -64(%edx,%ecx), %eax
  leal  -128(%edx,%ecx), %ecx
  movl  %ebx, %edi
  sall  $6, %edi
  subl  %edi, %ecx
  fstpl (%esp)
  .p2align 4,,10
  .p2align 3
.L3:
  fld %st(5)
  fmull 8(%eax)
  subl  $64, %eax
  faddl 64(%eax)
  fldt  8(%esp)
  fmull 80(%eax)
  faddp %st, %st(1)
  fldt  24(%esp)
  fmull 88(%eax)
  faddp %st, %st(1)
  fld %st(5)
  fmull 96(%eax)
  faddp %st, %st(1)
  fld %st(4)
  fmull 104(%eax)
  faddp %st, %st(1)
  fld %st(3)
  fmull 112(%eax)
  faddp %st, %st(1)
  fld %st(2)
  fmull 120(%eax)
  faddp %st, %st(1)
  fldl  (%esp)
  fmul  %st(2), %st
  faddp %st, %st(1)
  fstpl (%esp)
  cmpl  %eax, %ecx
  jne .L3
  fstp  %st(0)
  fstp  %st(0)
  fstp  %st(0)
  fstp  %st(0)
  fstp  %st(0)
  fldl  (%esp)
  negl  %ebx
  leal  -9(%esi,%ebx,8), %eax
  jmp .L2
  .p2align 4,,10
  .p2align 3
```

```
.L10:
  fstp  %st(4)
  fstp  %st(0)
  fstp  %st(0)
  fstp  %st(0)
  .p2align 4,,10
  .p2align 3
.L2:
  testl %eax, %eax
  js  .L11
  .p2align 4,,10
  .p2align 3
.L5:
  fmul  %st(1), %st
  faddl (%edx,%eax,8)
  subl  $1, %eax
  fstpl (%esp)
  fldl  (%esp)
  cmpl  $-1, %eax
  jne .L5
  fstp  %st(1)
  jmp .L1
  .p2align 4,,10
  .p2align 3
.L11:
  fstp  %st(1)
.L1:
  addl  $40, %esp
  .cfi_def_cfa_offset 16
  popl  %ebx
  .cfi_restore 3
  .cfi_def_cfa_offset 12
  popl  %esi
  .cfi_restore 6
  .cfi_def_cfa_offset 8
  popl  %edi
  .cfi_restore 7
  .cfi_def_cfa_offset 4
  ret
  .cfi_endproc
```

opt_poly (64-bit)

```
poly_opt:
.LFB0:
  .cfi_startproc
  endbr64
  movapd  %xmm0, %xmm3
  movapd  %xmm0, %xmm4
  movapd  %xmm0, %xmm5
  mulsd %xmm0, %xmm3
  movapd  %xmm0, %xmm6
  movapd  %xmm0, %xmm7
  movapd  %xmm0, %xmm8
  leaq  0(,%rsi,8), %r8
  movapd  %xmm0, %xmm9
  leaq  (%rdi,%r8), %rax
  leaq  -1(%rsi), %rdx
  movapd  %xmm0, %xmm2
  movsd (%rax), %xmm0
  mulsd %xmm3, %xmm4
  mulsd %xmm4, %xmm5
  mulsd %xmm5, %xmm6
  mulsd %xmm6, %xmm7
  mulsd %xmm7, %xmm8
  mulsd %xmm8, %xmm9
  cmpq  $6, %rdx
  jle .L2
  leaq  -8(%rsi), %rcx
  leaq  -64(%rdi,%r8), %rdx
  shrq  $3, %rcx
  movq  %rcx, %r8
  salq  $6, %r8
  subq  %r8, %rdx
  .p2align 4,,10
  .p2align 3
.L3:
  movsd -48(%rax), %xmm10
  movapd  %xmm0, %xmm1
  movsd -56(%rax), %xmm0
  subq  $64, %rax
  mulsd %xmm9, %xmm1
  mulsd %xmm3, %xmm10
  mulsd %xmm2, %xmm0
  addsd (%rax), %xmm0
  addsd %xmm10, %xmm0
  movsd 24(%rax), %xmm10
  mulsd %xmm4, %xmm10
  addsd %xmm10, %xmm0
  movsd 32(%rax), %xmm10
  mulsd %xmm5, %xmm10
  addsd %xmm10, %xmm0
  movsd 40(%rax), %xmm10
  mulsd %xmm6, %xmm10
  addsd %xmm10, %xmm0
  movsd 48(%rax), %xmm10
  mulsd %xmm7, %xmm10
```

```
  addsd %xmm10, %xmm0
  movsd 56(%rax), %xmm10
  mulsd %xmm8, %xmm10
  addsd %xmm10, %xmm0
  addsd %xmm1, %xmm0
  cmpq  %rax, %rdx
  jne .L3
  negq  %rcx
  leaq  -9(%rsi,%rcx,8), %rdx
.L2:
  testq %rdx, %rdx
  js  .L1
  .p2align 4,,10
  .p2align 3
.L5:
  mulsd %xmm2, %xmm0
  addsd (%rdi,%rdx,8), %xmm0
  subq  $1, %rdx
  cmpq  $-1, %rdx
  jne .L5
.L1:
  ret
  .cfi_endproc
```

Comparison

Both version of the algorithm was compiled with the `-O2` flag for `gcc`.

The first thing that jumps out in the difference between the two is the lack of `xmm` registers being used in the 32-bit version. Instead, the 32-bit assembly code uses the stack to store values, especially in the case of my code where I pre-compute powers of x and store them locally in the function before using them.

Aside from that, memory access is much more prevalent in the 32-bit version compared to the 64-bit version. There are a lot more operations moving element to and from the stack in the 32-bit version, therefore likely increasing the number of cycles.