目录

第 17 章 AVX 程序设计

高级向量扩展（Advanced Vector Extensions，AVX）是在 MMX、SSE 序列之后于 2011 年出现的 SIMD 增强版。2013 年，Intel 发布了 AVX2，2016 年推出了 AVX-512。本章简单介绍 x86-AVX 技术的新增处理能力、运行环境、指令系统，给出了 AVX 编程示例。

17．1 AVX 技术简介

1、AVX 概览

2011 年，Intel 在微处理器架构 Sandy Bridge 中引入了第一代 AVX，它将 SSE 中的浮点数运算能力从 128 位扩展到了 256 位。另一个很大的变化是支持了新的三目运算符指令，简化了汇编语言程序的编写。采用新的指令系统，可以实现 128 位的组合整数、128 位和 256 位的组合浮点数运算。在 Intel 酷睿（Core）处理器 i3、i5、i7 系列中使用了 Sandy Bridge 微架构。2012 年，Intel 发布了更新版的微处理结构 Ivy Bridge，引入了半精度浮点数变换指令。

2013 年，Intel 发布了新的微处理器架构 Haswell，引入了 AVX2。AVX2 支持 256 位的组合整型的运算，增强了数据广播、混合和排列指令。引入了新的向量索引寻址模式，增加了从不连续的地址空间载入数据的能力、乘法与加法融合运算（Fused Multiply Add，FMA）能力、位操作以及不带标志位的循环和位移能力。面向工作站和服务器的 Xeon E3 v3 采用了 Haswell 微架构。

2016 年，Intel 发布了 AVX-512，在 Intel 酷睿 i7 和酷睿 i9 等 CPU 中都支持了 AVX-512。这也是在深度学习火热的背景下，面对大规模的数据计算问题，Intel 公司不甘落后，向 NVIDIA（英伟达）的 GPU 发出挑战。AVX-512 支持 512 位的组合整数运算、512 位的组合单/双精度度浮点运算。

2、AVX 中的数据寄存器

支持 AVX 技术的处理器中有 8 个 256 位的寄存器，名为 ymm0～ymm7。可以在指令中直接使用这些寄存器的名字，即采用寄存器寻址方式访问组合整数、组合浮点数和标量浮点数。但是这些寄存器不能用于寄存器间接寻址、变址寻址和基址加变址寻址，即不能用于寻址内存中的操作数。

YMM 寄存器的低 128 位可以看成是一个 XMM 寄存器，AVX 指令中既可以使用 YMM 做为操作数也可以使用 XMM 作为操作数。在使用 SSE 指令时，对应 XMM 寄存器的 YMM 寄存器的高 128 位保持不变；但是使用 AVX 指令时，若目的操作数为 XMM 寄存器，则对应的 YMM 的高 128 位被设置为 0。

3、半精度浮点数

半精度浮点数用 16 个二进制位来存储，最高位为符号位，之后是指数部分（5 位）、有

效数字部分（10 位）。半精度浮点数无法进行加、减、乘、除等运算，主要是用来节约存储空间的。

4、乘法与加法混合运算

设要计算 r = x*y+z，用以前的方法处理，处理器在执行完 x*y 后，会进行一个舍入操作，然后再做加法，进行第二次舍入操作。采用乘法与加法混合运算（Fused Multiply Add，FMA）指令，不会对乘法的结果做舍入处理，只有在得到最后加法的结果后，才会做舍入操作。这样使用 FMA 指令可以提高乘法累加（如向量内积）运算的性能和精度。

17．2 AVX 指令简介

AVX 在指令编码模式中采用了一种新的前缀(VEX)。大多数 AVX 指令采用了三目运算符的格式：“VOP DesOp, SrcOp1,SrcOp2”,SrcOp1 和 SrcOp2 为源操作数地址，DesOp 为目的操作数地址。例如，对于 SSE 中的指令 addps xmm1,xmm2/m128,可以等价的写成 vaddps xmm1,xmm2,xmm3/m128。AVX 指令集大致可分为三类：一是用新的表示方法但功能等效于 SSE 指令的升级版本指令；二是新引入的指令；三是功能扩展指令，包括半精度浮点数变换、FMA 指令和新的通用寄存器指令。本节主要介绍一些新增加的指令和功能扩展指令。

17.2.1 新指令

AVX 新指令包括数据广播指令、提取和插入指令、掩码移动指令、排列指令、变长移位指令、数据收集指令等。

1、数据广播

将一个数据（整型字节、字、双字、四字、单精度浮点数、双精度浮点数、组合的 128 位数据）拷贝到目的操作数中的各个子元素中。指令有 vpbroadcastb、vpbroadcastw、vpbroadcastd 、 vpbroadcastq 、 vbroadcastss 、 vbroadcastsd 、 vbroadcasti128 、vbroadcastf128。

2、数据提取和插入

在 YMM 寄存器和 XMM 寄存器之间或者内存之间的部分数据的传送，从或向 YMM 的指定位置提取或插入数据。指令有 vextracti128，vextractf128、vinserti128、vinsertf128。

3、数据掩码移动

根据掩码的值，决定目的操作数中的某一位置是否从源操作数的对应位置拷贝数据，对于未拷贝数据的位置，元素置为 0。指令有：vmaskmovps、vmaskmovpd、vpmaskmovd、vpmaskmovq。

4、数据排列

按照指定的顺序，对于源操作数中的各个元素进行重排，结果存入目的寄存器中。数据重排指令有:vpermd、vpermq、vperms、vpermpd、vpermps、vpermilpd、vpermilps、vperm2i128、

vperm2f128。

5、变长移动

在移位指令中，对于组合数据中的各个元素分别采用不同的移动位数，即变长移位。指令有：vpsllvd、vpsllvq、vpsravd、vpsrlvd、vpsrlvq。

6、数据收集

从内存的不同位置拷贝数据到谜底寄存器中。内存位置采用了一种称为 VSIB 的特殊寻址方式。VSIB（Vector Scale Index Base）中含有 4 个元素：

基址(Base)：数组的首地址，放在一个通用的寄存器中；

比例因子(Scale)：数组元素的大小，其值为 1,2,4 或 8；

索引(Index)：数组元素的下标；

位移量(Displacement)：可选项，用来指定距离数据起始位置的偏移量。

在数据收集中，还有各个元素的条件拷贝控制掩码。对于掩码为 1 的元素拷贝，反之则保留原元素不变。

数据收集指令有：vgatherdpd、vgatherqpd、vgatherdps、vgatherqps、vgatherdd、vgatherqd、vgatherdq、vgatherqq。

17.2.2 功能扩展指令

功能扩展指令包括：乘法与加法融合运算（Fused Multiply Add，FMA）、通用寄存器指令、半精度浮点数转换指令等。

1、FMA 指令

乘法与加法融合指令可以分成几个小类，包括组合浮点或者标量浮点的乘加融合、乘减融合、组合整型的乘加融合、乘减融合等。

① vfmadd***pd|ps|sd|ss op1, op2, op3 浮点乘加融合

pd 和 ps 分别为组合双精度和组合单精度浮点;sd 和 ss 是标量双和单精度浮点之意。

*** 是一个由 1、2、3 三位数的组合，指明在运算表达式中三个 op 出现的顺序。例如,132 代表 op1 * op3 + op2。op1、op2 是 128 位或者 256 位的 XMM 寄存器或 YMM 寄存器；op3 可以是 XMM 寄存器、YMM 寄存器或者内存单元。

浮点乘加融合的具体指令有：vfmadd132pd、vfmadd132ps、vfmadd132sd、vfmadd132ss、vfmadd213pd、vfmadd231pd 等等。

② vfmsub***pd|ps|sd|ss op1, op2, op3 浮点乘减融合

与浮点乘加融合相对应，只是将 add 替换为 sub，再乘法之后做减法运算。

③ vfmaddsub***pd|ps op1, op2, op3 浮点乘法、加法、减法融合

只用于组合型数据的运算，对于组数数据中的序号为奇数的元素做加法，对序号为偶数的元素做减法运算。

④ vfmsubadd***pd|ps op1, op2, op3 浮点乘法、减法、加法融合

只用于组合型数据的运算，与 vfmaddsub***pd|ps 类似，只是对奇元素做减法，偶元素做加法运算。

⑤ vfnmadd***pd|ps|sd|ss op1, op2, op3 浮点乘的相反数加法融合

在乘法运算后，对乘的结果取相反数，然后再做加法。

⑥ vfnmsub***pd|ps|sd|ss op1, op2, op3 浮点乘的相反数减法融合

在乘法运算后，对乘的结果取相反数，然后再做减法。

2、通用寄存器指令

增强了一些涉及到通用寄存器的指令，下面仅列出了其中的一部分。

① andn desop, srcop1,srcop2：第一个源操作数（srcop1）求反后与第二个源操作数（srcop2）进行逻辑与操作，结果保存在目的操作数中。目的操作数和第一个源操作数应为通用寄存器，第二个源操作数可为通用寄存器或内存单元。

② sarx、shlx、shrx：算术右移、逻辑左移、逻辑右移

③ pextr：从第一个源操作数的指定位置开始提取指定长度的二进制位，存放到目的操作数中。

④ blsi ：提取源操作数的最低位，存放到目的操作数的最低位，目的操作数的其他为置为 0。

⑤ lzcnt：统计源操作数中前导 0 的个数（CouNT the number of Leading Zero bits）

17．3 AVX 编程示例

AVX 的指令很多，对于一个初学者要掌握这些指令还是有一定困难的。同样，我们可以写一个 C 语言程序，在编译时，在代码生成的“启用增强指令集”的选项中选择“高级矢量扩展 (/arch:AVX)”，则会生成使用 AVX 指令的代码。

1、C 语言程序示例

我们同样使用第 16 章中的 C 语言程序示例，实现两个浮点数相加，然后显示结果。

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    float x, y, z;
    x = 3.14;
    y = 5.701;
    z = x + y;
    printf("%f\n", z);
    return 0;
}
```

使用“高级矢量扩展 (/arch:AVX)”后生成的程序的反汇编程序片段如下。

```
    x = 3.14;
00421828  vmovss      xmm0,dword ptr [__real@4048f5c3 (0427B34h)]
```

```
00421830  vmovss       dword ptr [x],xmm0
    y = 5.701;
00421835  vmovss       xmm0,dword ptr [__real@40b66e98 (0427B38h)]
0042183D  vmovss       dword ptr [y],xmm0
    z = x + y;
00421842  vmovss       xmm0,dword ptr [x]
00421847  vaddss       xmm0,xmm0,dword ptr [y]
0042184C  vmovss       dword ptr [z],xmm0
    printf("%f\n", z);
00421851  vcvtss2sd    xmm0,xmm0,dword ptr [z]
00421856  sub          esp,8
00421859  vmovsd       qword ptr [esp],xmm0
0042185E  push         offset string "%f\n" (0427B30h)
00421863  call         _printf (0421046h)
00421868  add          esp,0Ch
```

在调试时,在寄存器窗口中可以看到 XMM0 低 4 字节内容与 YMM0 的第 4 字节内容相同。由于 YMM 的位数很多，在寄存器窗口也将其拆成了 4 个 64 位的显示形式。

2、汇编语言程序示例

对于上面的例子，用汇编语言实现相同功能的代码如下。

```
.686P
.XMM
.model flat, c
  ExitProcess proto stdcall :dword
  printf      proto  :VARARG
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
.data
  lpFmt db  "%f",0ah, 0dh, 0
  x  real4  3.14
  y  real4  5.701
  z  real4  0.0
.stack 200
.code
main  proc
   vmovss xmm0, x
```

```
    vaddss xmm0,xmm0, dword ptr y
    vmovss z, xmm0
    cvtss2sd xmm0,z
    sub  esp,8
    vmovsd   qword ptr [esp], xmm0
    invoke printf,offset lpFmt
    invoke Exitprocess, 0
main  endp
end
```

注意，单精度浮点数和双精度浮点数之间要实现类型 C 语言的赋值功能时，要使用相应的类型转换指令，若只是简单地在它们之间传送，实现的只是内容的拷贝。显然，对于一个浮点数，用单精度来表示与双精度表示的结果并不相同，内容的拷贝无法实现类型的转换功能。

另外，与 C 语言编程中使用 MMX、SSE 技术一样，在 C 语言编程中可以使用 AVX 函数，以提高大规模数据运算的能力。具体的函数可以参考头文件 immintrin.h。

在编程时，对 CPU 是否支持采用的指令集可以进行判断。使用 CPUID 指令可以获得 CPU 的多种信息，进而判断是否支持相应的指令集。

习题 17

17.1　AVX 中有哪 8 个 256 位的寄存器？

17.2　设有变量

```
x       dw    70H,   0FFA0H,   50H,   50H,   0F0H, 0F0H, 0F000H, 0F0H
y       dw   0A0H,   0070H,   30H,   0F0H,   01H,   20H,   8001H, 0F0H
z       dw   8 DUP(0)
```

编写程序，完成 z=x+y，x、y 和 z 都看成为向量。要求分别用下面两种规则实现：

① 使用 SSE 指令，实现无符号字饱和字加法；

② 使用 SSE 指令，实现字环绕加法。

上机实践 17

17.1　用 C 语言编写程序，实现两个矩阵的乘法。对矩阵相乘的部分进行计时。

假设矩阵中元素类型为 short，矩阵行列数都是 8 的倍数。

(1) 用传统方法（即逐个元素运算）实现

(2) 用 SSE 打包运算方法（参见 emmintrin.h 中的函数）

17.2 功能与上机实践 17.1 相同，即实现矩阵的乘法，但矩阵中的行、列数为任意正整数。