并行复习3 SIMD编程

SIMD编程（多媒体扩展编程）

SIMD：单指令多数据流

SIMD编程的问题(打包解包、对齐开销、控制流开销)

SIMD编程的额外开销

打包/解包数据的开销：重排数据使之连续

打包：将数据拷贝到连续的内存区域，然后读进向量寄存器

解包：将数据从向量寄存器拷贝回内存中

对齐：调整数据访问，使之对齐

- 对齐的内存访问：

地址总是向量长度的倍数（是16字节的整数倍）

因此如果存储的数据不再向量长度的倍数位置, 则会额外读取一段, 在进行合并. 造成额外开销

- 未对齐的内存访问：

地址不是16字节的整数倍

静态对齐：对未对齐的读操作，做两次相邻的对齐读操作，然后进行合并。

有时硬件会帮你做, 但仍然会产生多次内存操作

动态对齐：合并点在运行时计算

- 对齐问题小结

最坏情况需要计算地址，动态对齐。

编译器（程序员）可以分析确认对齐。

可以调整算法，先串行处理到对齐边界，然后进行SIMD计算。

有时对齐开销会完全抵消SIMD的并行收益。

控制流可能要求执行所有路径

- 如果程序中有控制流变化, 就需要所有路径都执行，因此造成额外开销。
- 如何改进：假定所有控制流路径执行频率都不同，应该针对频率最高的路径优化代码，其他路径按默认方式执行。

- 小结：

当前的商用编译器不太可能支持控制流优化

一般观点，当SIMD存在控制流问题时，将不是一个好的编程模型

一些情况下还是可能加速的

SIMD编程

SIMD（Single Instruction Multiple Data）是单指令多数据技术，目前Intel处理器支持的SIMD技术包括MMX，SSE，AVX。

SSE和AVX编程的基本知识

支持多少位的计算

SSE编程支持一次计算几个双精度浮点数

SSE（Stream SIMD Extentions，数据流单指令多数据扩展）

用一个控制器对一组数据（又称“数据向量”）中的每一个分别执行相同的操作来实现空间上的并行性。

SSE指令集：8个128位向量寄存器

4个单精度浮点数，2个双精度浮点数

SSE编程 C/C++

SSE指令对应C/C++ intrinsic

- 数据类型

__m128:float

__m128d:double

__123i:integer

- 数据移动和初始化

☒ _mm_load_ps, _mm_loadu_ps, _mm_load_pd, _mm_loadu_pd, ...

从内存加载4个单精度浮点数到一个128位的XMM寄存器;与 `_mm_load_ps` 类似，但是不要求内存地址对齐;从内存加载2个双精度浮点数到一个128位的XMM寄存器;与 `_mm_load_pd` 类似，但是不要求内存地址对齐.

ps:打包的单精度浮点数

u:非对齐

pd:打包的双精度浮点数

☒ _mm_store_ps, …

将一个128位的XMM寄存器中的4个单精度浮点数存储到内存。

☒ _mm_setzero_ps

创建一个所有元素为零的128位XMM寄存器

☒ _mm_loadl_pd, _mm_loadh_pd

从内存加载一个双精度浮点数到一个128位XMM寄存器的低64位;从内存加载一个双精度浮点数到一个128位XMM寄存器的高64位

l:low 低64位

h:high 高64位

☒ _mm_storel_pd, _mm_storeh_pd

将一个128位XMM寄存器的低64位双精度浮点数存储到内存；将一个128位XMM寄存器的高64位双精度浮点数存储到内存

☒ _mm_set_ps, _mm_set1_ss, _mm_setzero_ss

创建一个包含4个单精度浮点数的128位XMM寄存器；创建一个所有元素都相同的128位XMM寄存器，其中只有低32位包含单精度浮点数；创建一个所有元素为零的128位XMM寄存器，其中只有低32位包含单精度浮点数。

- 算数intrinsics

☒ _mm_add_ss, _mm_add_ps, …

将两个单精度浮点数相加并将结果储存在一个标量单精度浮点数寄存器中

将两组打包的（通常是4个）单精度浮点数相加，每组中的元素分别相加，结果存在一个XMM寄存器中

☒ _mm_add_pd, _mm_mul_pd

将两组打包的双精度浮点数（通常是2个）相加，每组中的元素分别相加，结果存储在一个XMM寄存器中

将两组打包的双精度浮点数相乘，每组中的元素分别相乘，结果存储在一个XMM寄存器中。

☒ _mm_hadd_ps

将两组打包的单精度浮点数（通常是4个）进行水平相加

- 预取intrinsics

_mm_prefetch(char const *a, int sel)

intrinsic函数的命名

分三部分组成

- Intrinsic函数的命名有一定的规律，一个Intrinsic通常由3部分构成：
  - 第一部分为前缀_mm，表示是SSE指令集对应的Intrinsic函数。_mm256或_mm512是AVX,AVX-512指令集的Intrinsic函数前缀，这里只讨论SSE故略去不作说明。
  - 第二部分为对应的指令的操作，如_add，_mul，_load等，有些操作可能会有修饰符，如loadu将16位未对齐的操作数加载到寄存器中。
  - 第三部分为操作的对象名及数据类型，_ps packed操作所有的单精度浮点数；_pd packed操作所有的双精度浮点数；_pixx（xx为长度，可以是8，16，32，64）packed操作所有的xx位有符号整数，使用的寄存器长度为64位；_epixx（xx为长度）packed操作所有的xx位的有符号整数，使用的寄存器长度为128位；_epuxx packed操作所有的xx位的无符号整数；_ss操作第一个单精度浮点数（标量）。
- 将这三部分组合就是一个Intrinsic函数，如_mm_mul_epi32对参数中所有的32位有符号整数进行乘法运算。

65

矩阵乘法

向量编程存在什么样的问题

(打包解包、对齐开销、控制流开销)这三种开销比较高

具体编程的变量怎么命名（分三个字段