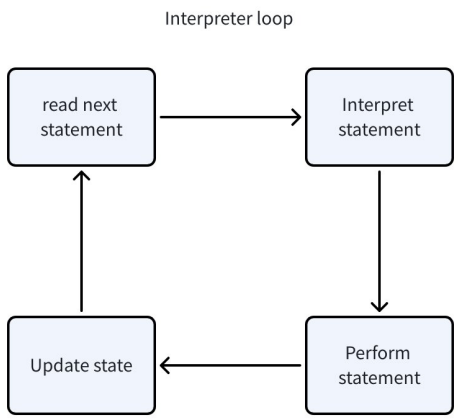
[MIT 6.172] 1-Introduction and Matrix Multiplication

🖊️ Why python is so slow & C is so fast?

因为Python是解释型的语言，而C是为机器直接编译的语言。Java在两者之间，是因为java被编译为字节码，然后被解释，然后即时编译。

🖊️ Interpreters are versatile, but slow

- 解释器读取、解释并执行每条程序语句，同时更新机器状态。
- 解释器能够轻松支持高级编程特性（如动态代码修改），但会以牺牲性能为代价



🖊️ JIT Compilation

- 即时（JIT）编译器能够挽回因解释执行而损失的部分性能。
- 代码首次执行时，会以解释方式运行。
- 运行时系统会跟踪不同代码片段的执行频率。
- 一旦某段代码的执行频率足够高，就会实时编译为机器码。
- 该代码后续执行时，会使用效率更高的编译版本。

🖊️ Performances of different orders

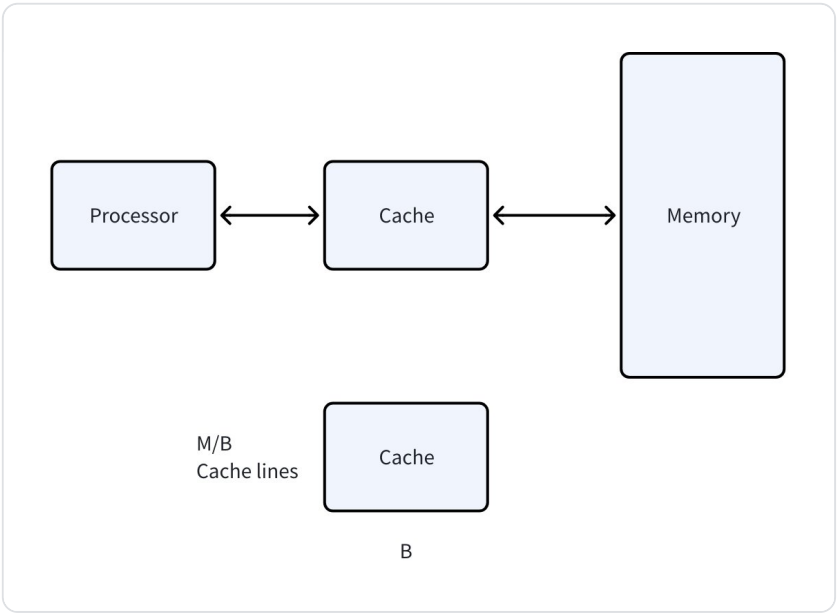
每个处理器以连续的块（称为“缓存行”）为单位读写主存。

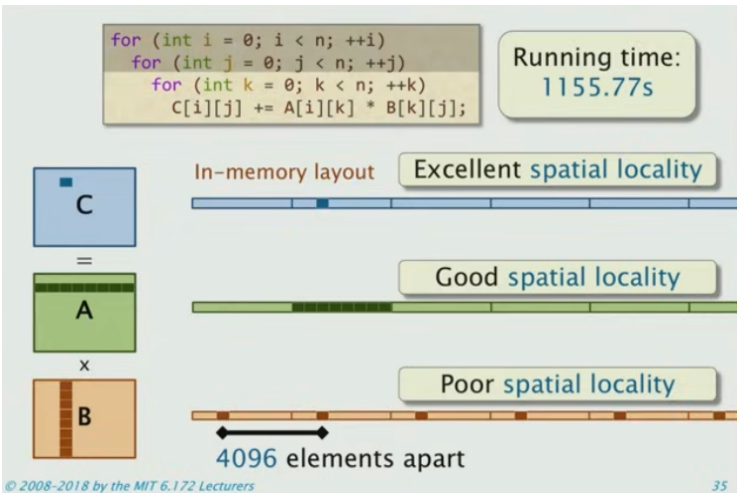
- 之前访问过的缓存行会存储在一个更小的内存（称为“缓存”）中，该缓存靠近处理器。
- 缓存命中（访问缓存中的数据）速度很快。
- 缓存未命中（访问不在缓存中的数据）速度很慢。

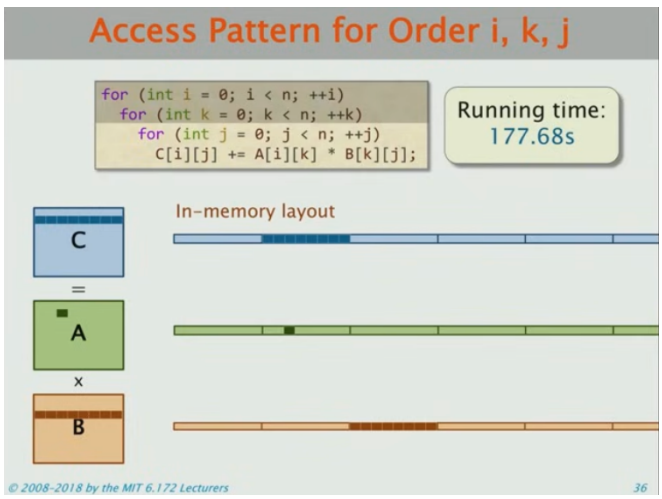
检查工具：Cachegrind cache simulator

`valgrind --tool=cachegrind ./mm`

Loop order (outer to inner)	Running Time(s)
i,j,k	1155.77
i,k,j	177.68
j,i,k	1080.61
j,k,i	3056.63
k,i,j	179.21
k,j,i	3032.82







🖊 Complier Optimization

- Clang提供一系列优化开关。可以向编译器制定一个开关，要求它进行优化
- Clang还支持特殊目的的优化
  - -Os：限制代码大小

- -Og：用于调试目的

优化级别	含义	时间	解释
-O0	不优化	177.54	
-O1	优化	66.24	
-O2	进一步优化	54.63	
-O3	更深度优化	55.58	

🖊 Parallel Loops

- 在c语言中引入库 `` `#include <cilk.h>` ``
- 使用 `` `cilk_for` `` 循环允许循环的所有迭代并行执行

🖊 Hardware Caches, Revisited

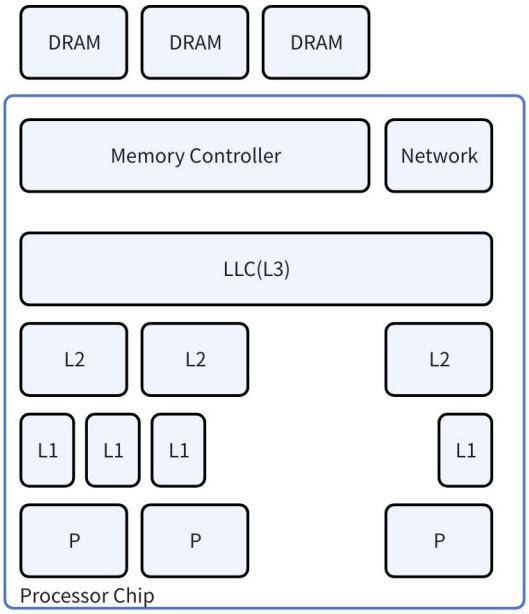
- Idea：重构计算过程中，以尽可能的复用缓存中的数据
- 缓存未命Cache misses中速度慢，缓存命中Cache hits速度快
- 尽量通过复用已咋缓存中的数据，充分利用缓存
- 使用分块矩阵乘法
  - s：Tuning parameters 调优参数

代码块

```
cilk_for (int ih=0;ih<n;ih+=s) //best for s:32
    cilk_for (int jh=0;jh<n;jh+=s)
        for (int kh=0;kh<n;kh+=s)
            for(int il=0;il<s;++il)
                for(int kl=0;kl<s;++kl)
                    for(int jl=0;jl<s;++jl)
                        C[ih+il][jh+jl]+=A[ih+jh][kh+kl]+B[kh+kl][jh+jl];
```

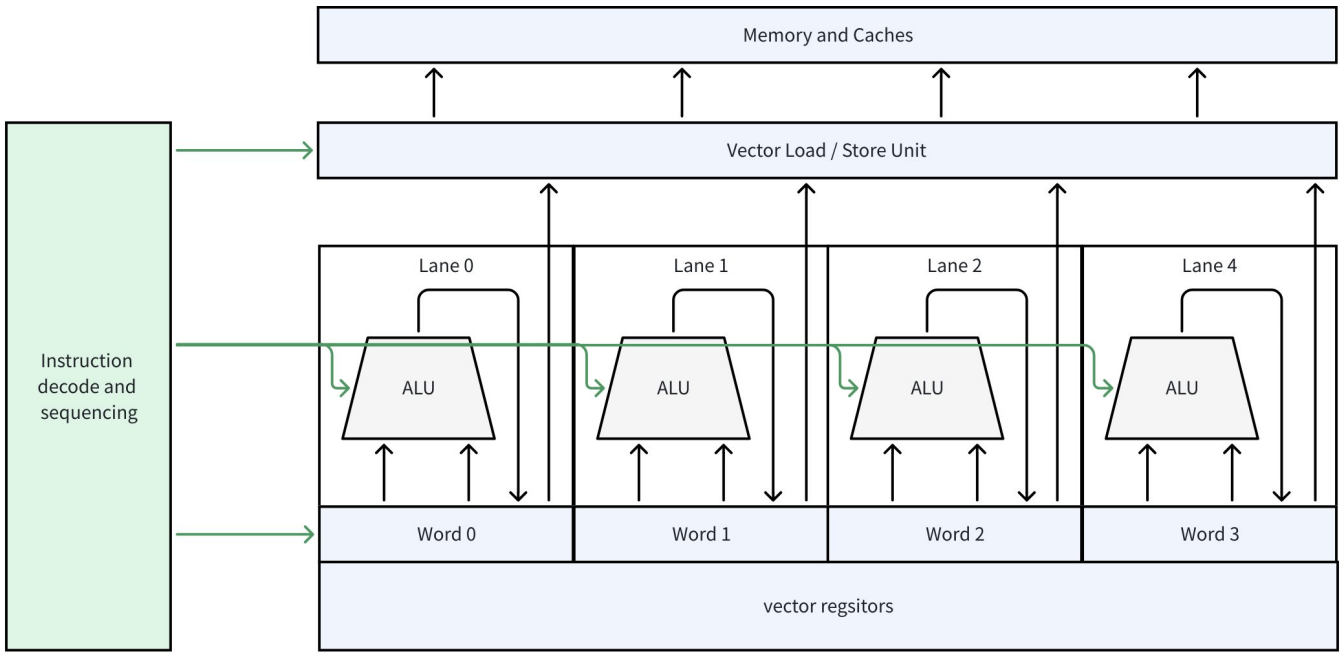
🖊 Multicore Cache Hierachy

- 可以增加为2个调优参数：s和t
- Idea：同时为两个矩阵中的块进行分块处理



🖊️ Vector Hardware

- 现代微处理器集成了向量硬件，以单指令流、多数据流（SIMD）的方式处理数据
  - SIMD是一种并行技术，能让处理器一条指令同时对多个数据进行操作，提升运算效率
- Clang/LLVM 在优化级别为-O2或更高时，会自动使用向量指令进行编译，生成向量化报告
- 可以使用如下编译器标志 complier flags，指示编译器使用现代向量指令
  - `-mavx` 使用英特尔AVX2向量指令
  - `-mavx2` 使用英特尔AVX2向量指令
  - `-mfma` 使用融合乘加（fused multiply -add）向量指令
  - `-march=<字符串>` 使用指定架构支持的所有指令
  - `-march=native` 使用执行编译操作的机器架构所支持的所有指令
    - 由于浮点数运算存在限制，如果要让这些向量化标志生效，可能需要额外的标志，比如 `-ffst -math`



AVX Intrinsic Instructions

Intel provides C-style functions, called *intrinsic instructions*, that provide direct access to hardware vector operations:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/

intel Intrinsics Guide

The Intel Intrinsics Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code.

Technologies

- [ ] MMX
- [ ] SSE
- [ ] SSE2
- [ ] SSE3
- [ ] SSSE3
- [ ] SSE4.1
- [ ] SSE4.2
- [x] AVX
- [x] AVX2
- [x] FMA
- [ ] AVX-512
- [ ] KNC
- [ ] SVML
- [ ] Other

Categories

- [ ] Application-Targeted

_mm_search ?

```
__m256i _mm256_abs_epi16 (__m256i a)                          vpabsw
__m256i _mm256_abs_epi32 (__m256i a)                          vpabsd
__m256i _mm256_abs_epi8 (__m256i a)                           vpabsb
__m256i _mm256_add_epi16 (__m256i a, __m256i b)               vpaddw
__m256i _mm256_add_epi32 (__m256i a, __m256i b)               vpaddd
__m256i _mm256_add_epi64 (__m256i a, __m256i b)               vpaddq
__m256i _mm256_add_epi8 (__m256i a, __m256i b)                vpaddb
__m256d _mm256_add_pd (__m256d a, __m256d b)                  vaddpd
__m256 _mm256_add_ps (__m256 a, __m256 b)                     vaddps
__m256i _mm256_adds_epi16 (__m256i a, __m256i b)              vpaddsw
__m256i _mm256_adds_epi8 (__m256i a, __m256i b)               vpaddsb
__m256i _mm256_adds_epu16 (__m256i a, __m256i b)              vpaddusw
```

还有一整本