提升指令缓存的命中率——计算机组成原理

张岩峰，8208201030

中南大学计算机学院

摘　要： 代码都是由 CPU 跑起来的，我们代码写的好与坏就决定了 CPU 的执行效率，特别是在编写计算密集型的程序时，更要注重 CPU 的执行效率，否则将会大大影响系统性能。CPU 内部嵌入了 CPU Cache（高速缓存），它的存储容量很小，但是离 CPU 核心很近，所以缓存的读写速度是极快的，那么如果 CPU 运算时，直接从 CPU Cache 读取数据，而不是从内存的话，运算速度就会很快。本文试图向读者介绍什么是 CPU Cache，它是如何工作的，又该怎样写出让 CPU 执行更快的代码。

关键词： 计算机；计算机组成原理；中央处理器；高速缓存；动态分支预测；

Enhancement of Instruction Cache Hit Rate — Principles of Computer Composition

Bruce Chang， 8208201030

Central South University, School of Computer Science and Engineering

Abstract: The CPU is embedded with a CPU Cache, which has a small storage capacity but is very close to the CPU core, so the read/write speed of the cache is very fast. If the CPU reads data directly from the CPU Cache instead of from memory, then the computation speed will be very fast. This article tries to introduce the reader to what CPU Cache is, how it works, and how to write code that makes the CPU execute faster.

Key Words: Computer; Principles of Computer Composition; CPU; Cache; Branch Predictor;

1　背景

正常来说，存储器的容量和性能应该伴随着 CPU 的速度和性能提升而提升，以匹配 CPU 的数据处理。但随着时间的推移，CPU 和存储器在性能上的发展差异越来越大，存储器在性能增长越来越跟不上 CPU 性能发展的需要。

2　意义

CPU 访问内存的速度，比访问 CPU Cache 的速度慢了 100 多倍，所以如果 CPU 所要操作的数据在 CPU Cache 中的话，这样将会带来很大的性能提升。访问的数据在 CPU Cache 中的话，意味着缓存命中，缓存命中率越高的话，代码的性能就会越好，CPU 也就跑的越快。

3　研究现状

最早期的计算机，在执行一段程序时，都是把硬盘中的数据加载到内存，然后CPU从内存中取出代码和数据执行，在把计算结果写入内存，最终输出结果。

随着程序运行越来越多，人们发现一个规律：内存中某个地址被访问后，短时间内还有可能继续访问这块地址。内存中的某个地址被访问后，它相邻的内存单元被访问的概率也很大。

这种规律被称为程序访问的局部性。

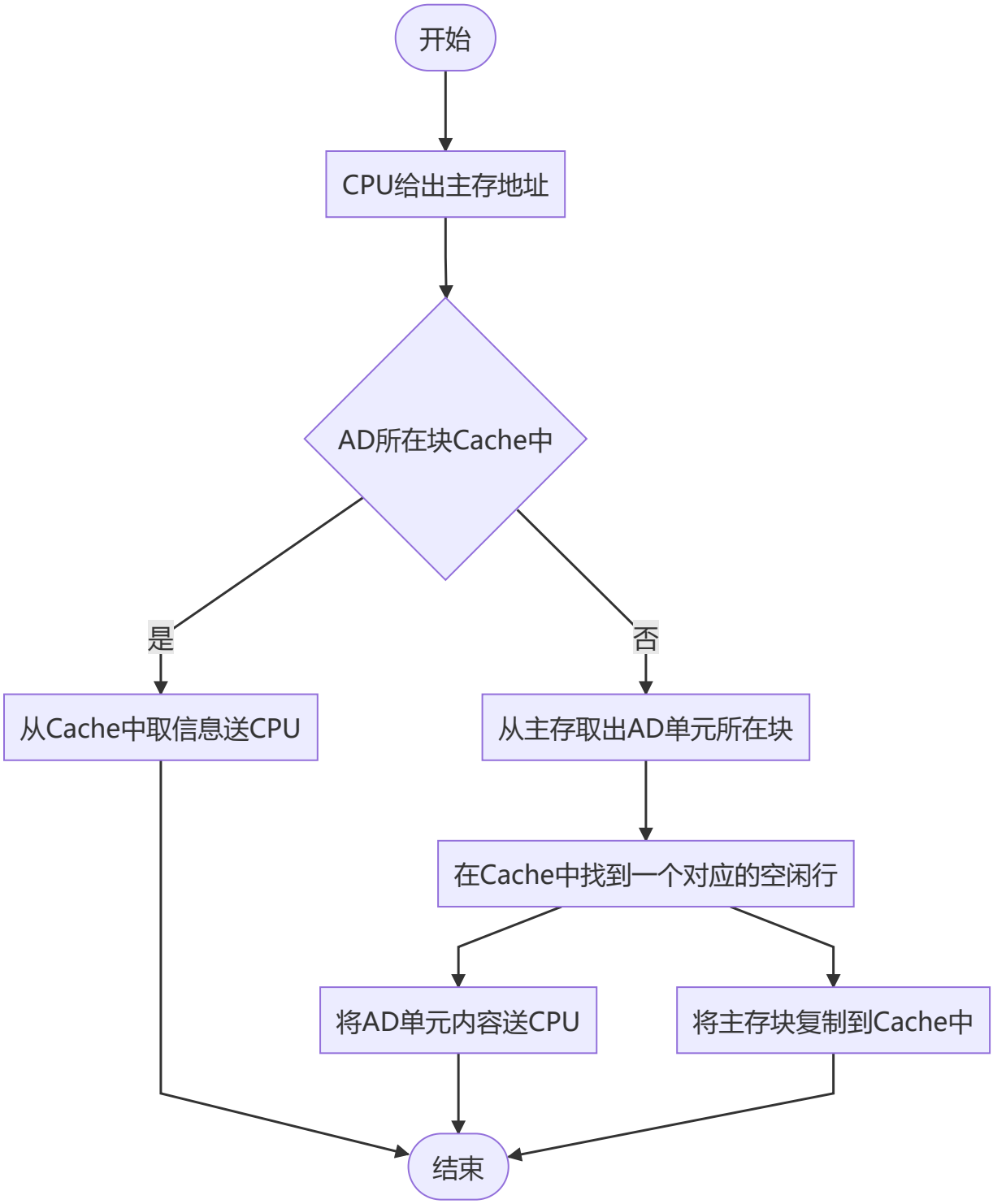
程序访问的局部性包含2种：

- 时间局部性：某个内存单元在较短时间内很可能被再次访问
- 空间局部性：某个内存单元被访问后相邻的内存单元较短时间内很可能被访问

出现这种情况的原因很简单，因为程序是指令和数据组成的，指令在内存中按顺序存放且地址连续，如果运行一段循环程序或调用一个方法，又或者再程序中遍历一个数组，都有可能符合上面提到的局部性原理。

在执行程序时，内存的某些单元很可能会经常的访问或写入，那可否在CPU和内存之间，加一个缓存，CPU在访问数据时，先看一下缓存中是否存在，如果有直接就读取缓存中的数据即可。如果缓存中不存在，再从内存中读取数据。事实证明利用这种方式，程序的运行效率会提高90%以上，这个缓存也叫做高速缓存Cache。

CPU对内存、高速缓存Cache进行数据访问的流程如下：



4 实验结果及分析

如何提升指令缓存的命中率?

提升数据的缓存命中率的方式，是按照内存布局顺序访问，那针对指令的缓存该如何提升呢?我们以一个例子来看看，有一个元素为 0 到 100 之间随机数字组成的一维数组。

接下来，对这个数组做两个操作:

- 第一个操作，循环遍历数组，把小于 50 的数组元素置为 0
- 第二个操作，将数组排序;

接下来我们采用两种不同的运行顺序，来观察是先遍历再排序速度快，还是先排序再遍历速度快:

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;

public class test {
    public static void main(String[] args) {
        //initialization
        int n = 1000;
        ArrayList<Integer> Array = new ArrayList<>();
        for(int i = 0; i < n; i++) {
            Array.add((int) (Math.random()*101));
        }

        long visitFirstStartTime = System.currentTimeMillis();

        //visit first
        for(int i = 0; i < n; i++) {
            if (Array.get(i) < 50) Array.set(i, 0);
        }

        Collections.sort(Array);

        Long visitFirstEndTime = System.currentTimeMillis();
        Long visitFirstElapsedTime = (visitFirstEndTime-visitFirstStartTime);
        System.out.println("先遍历再排序耗时: " + visitFirstElapsedTime + " 毫秒");

        //reInitialization
        for(int i = 0; i < n; i++) {
            Array.add((int) (Math.random()*101));
        }

        long sortFirstStartTime = System.currentTimeMillis();
```

```
        //sort first
        Collections.sort(Array);

        for(int i = 0; i < n; i++) {
            if (Array.get(i) < 50) Array.set(i, 0);
        }

        Long sortFirstEndTime = System.currentTimeMillis();
        Long sortFirstElapsedTime = (sortFirstEndTime-sortFirstStartTime);
        System.out.println("先排序再遍历耗时： " + sortFirstElapsedTime + " 毫秒");

    }
}
```

- 规模为 1k时

  先遍历再排序耗时: 6 毫秒
  先排序再遍历耗时: 4 毫秒

- 规模为 10k 时

  先遍历再排序耗时: 37 毫秒
  先排序再遍历耗时: 22 毫秒

- 规模为 100k 时

  先遍历再排序耗时: 103 毫秒
  先排序再遍历耗时: 88 毫秒

从上述结果可知，先排序后遍历耗时短于先遍历后排序，在回答为什么之前，我们先了解 CPU 分支预测器的概念。

对于 if 条件语句，意味着此时至少可以选择跳转到两段不同的指令执行，也就是 if 还是 else 中的指令。那么，如果分支预测可以预测到接下来要执行 if 里的指令，还是 else 指令的话，就可以「提前」把这些指令放在指令缓存中，这样 CPU 可以直接从 Cache 读取到指令，于是执行速度就会很快。

当数组中的元素是随机的，分支预测就无法有效工作，而当数组元素都是是顺序的，分支预测器会动态地根据历史命中数据对未来进行预测，这样命中率就会很高。

因此，先排序再遍历速度会更快，这是因为排序之后，数字是从小到大的，那么前几次循环命中 if < 50 的次数会比较多，于是分支预测就会缓存 if 里的 Array[i] $= 0$ 指令到 Cache 中，后续 CPU 执行该指令就只需要从 Cache 读取就好了。

5 结语

随着计算机技术的发展，CPU 与 内存的访问速度相差越来越多，如今差距已经高达好几百倍了，所以 CPU 内部嵌入了 CPU Cache 组件，作为内存与 CPU 之间的缓存层，CPU Cache 由于离 CPU 核心很近，所以访问速度也是非常快的，但由于所需材料成本比较高，它不像内存动辄几个 GB 大小，而是仅有几十 KB 到 MB 大小。

当 CPU 访问数据的时候，先是访问 CPU Cache，如果缓存命中的话，则直接返回数据，就不用每次都从内存读取速度了。因此，缓存命中率越高，代码的性能越好。

但需要注意的是，当 CPU 访问数据时，如果 CPU Cache 没有缓存该数据，则会从内存读取数据，但是并不是只读一个数据，而是一次性读取一块一块的数据存放到 CPU Cache 中，之后才会被 CPU 读取。

要想写出让 CPU 跑得更快的代码，就需要写出缓存命中率高的代码，CPU L1 Cache 分为数据缓存和指令缓存，因而需要分别提高它们的缓存命中率：

- 对于数据缓存，我们在遍历数据的时候，应该按照内存布局的顺序操作，这是因为 CPU Cache 是根据 CPU Cache Line 批量操作数据的，所以顺序地操作连续内存数据时，性能能得到有效的提升
- 对于指令缓存，有规律的条件分支语句能够让 CPU 的分支预测器发挥作用，进一步提高执行的效率

利用 CPU Cache 我们可以大幅提升 CPU 运行效率，相信在未来的发展下，计算机系统也会在不断优化完善的路上越走越远。