现代CPU使用的高速缓存

随着时间的推移以及现代计算机系统制造工艺的进步，处理器的速度与主存之间的速度差距呈现了指数级别的增长，而缓存及其层次结构的出现就是为了弥合这样大的速度差距。现代的计算机系统使用了存储器层次结构来组织存储器系统，这个结构类似于金字塔，从高层往低层走，存储设备变得更加慢、更便宜同时也更大。而CPU缓存是一种硬件，它通过将主存当中的经常使用的数据保存在更小更快的高速缓存当中来减少对速度相对较慢的主存中数据的访问时间。

高速缓存性能评估标准

对于高速缓存来说，有许多指标来衡量高速缓存的性能，而缓存的性能主要取决于缓存是否命中。当程序需要第`k + 1`层的某个数据对象`d`时，它会首先在当前存储在第`k`层的块中查找`d`，如果`d`刚好缓存在第`k`层当中，也就是缓存命中，否则就是发生了缓存不命中。缓存不命中又可以分为强制性不命中（缓存区为空时的访问不命中）、冲突不命中（由于缓存区中块数有限导致的数据放置冲突发生的不命中）以及容量不命中（工作集大小超过了缓存的大小导致的不命中），而在一个程序执行或者程序的一部分执行期间发生的内存引用不命中的比率就是不命中率，它是衡量高速缓存性能的标准之一。性能评估的标准还有命中时间，也就是从高速缓存传送一个字到CPU所需的时间，里面包括了组选择、行确认以及字选择的时间。不命中处罚也被用来评估高速缓存的性能，该处罚是由于缓存没有命中而需要从低一层次的高速缓存当中获取数据所需要花费的额外时间。

优化高速缓存的参数以及优化策略探讨

进一步地，可以探讨高速缓存相关参数对其性能产生的影响。首先是高速缓存的大小带来的影响，从一方面来说，较大的高速缓存可以提高命中率，但是另一方面又会增加命中时间，也因为这个原因，在存储器层次结构当中高一层次的高速缓存总是比低一层次的高速缓存要小。而对于给定的高速缓存大小，块的大小也会影响其性能，较大的块能够更好地利用程序当中可能存在的空间局部性，但也意味着高速缓存的行数会减少，这也会导致时间局部性利用率的下降。除此之外，缓存中较高的相联度会降低由于冲突不命中出现抖动的可能性，但是较高的相联程度会使得制造成本升高，也会增加命中时间以及不命中处罚，因为较高的相联程度意味着逻辑控制的复杂程度也上升了。

正如以上所说，要提高一个高速缓存的性能需要综合考量各种参数，进行对优化缓存成本以及性能的折中，不仅仅需要理论层次的推导，也要在运行实际程序当中进行大量的模拟。

高速缓存替换机制

当发生缓存不命中的情况时，此时第`k`层的缓存就会从第`k+1`层当中包含数据对象`d`的块，此时如果第`k`层的缓存已经满了，就会发生高速缓存的替换，而决定替换哪个块是由缓存的替换策略来控制的。

根据上述的高速缓存性能的影响因素，平均的存储参考时间可以用下列的公式进行表达

$$T = m * T_m + T_h + E$$

其中T是平均存储参考时间，m 代表不命中率，T_m代表不命中的处罚，T_h是命中时间。

在主要的缓存替换算法当中，最理想也是最有效的算法是`Belady`算法，这个算法总是丢弃在未来最长时间内不需要的信息从而得到缓存时间的最优结果。但是在实际情况当中无法在缓存的时候提前知道未来需要当前缓存当中数据的时间，所以实际无法实现该算法，但是可以使用该算法来比较缓存算法的有效性。第二种主要的算法就是LRU（最近最少被使用策略）算法，该算法的主要思想就是先覆盖最近最少被使用的数据，但是这种算法的代价比较高，因为总是需要为缓存线保存一个时间线去跟踪目标数据。除此之外还有一种算法和LRU比较类似，但是与LRU计算数据最近使用的时间不同，该算法是通过计算缓存中数据的使用频率，之后丢弃掉使用频率最低的数据。LRU的实现方式就是为缓存当中的每一个块分配一个计数器，每次出现对该数据块的引用就将计数器加一，当需要丢弃一个数据块时，系统将会丢弃计数器数量最少的数据块，很显然，这样的算法会导致新加入的数据块引用值总是比之前的数据块要少，所以总是会导致新加入的数据块被丢弃。而对缓存策略进行性能评估的方法主要可以分为三种：根据跟踪缓存当中数据块进行对缓存替换机制的仿真；还有根据综合模型模拟随机生成的请求模式，并且根据该请求模式进行分析。缓存替换类算法还有FIFO（使用队列来对缓存的替换行为进行模拟），FILO（使用栈来对缓存的替换过程进行模拟）以及RR（随机缓存），即当缓存需要丢弃数据块的时候随机地丢弃当中的数据块，这样的操作无论从操作的时间以及空间考虑都非常地简单，并且由于其机制的简便性，已经应用在ARM的处理器当中。

在论文《Optimum Caching versus LRU and LFU: Comparison and Combined Limited Look-Ahead Strategies》当中，作者提出了一种相对于LRU和LFU更加高效的优化算法——比较和组合的前瞻性策略（Comparison and Combined Limited Look-Ahead Strategies），该算法结合了上述的LRU和LFU缓存算法，通过模拟根据请求跟踪数据块的过程，以及独立参考模型 (IRM) 的前瞻程度来评估命中率带来的增益，并得出对观察到的缓存替换行为的分析确认。论文实验最后的结果表明该算法应用在视频流的缓存当中效果较好，因为视频流的缓存允许面对持续内容更新的时候根据新的需求来对替换策略进行部分修改。

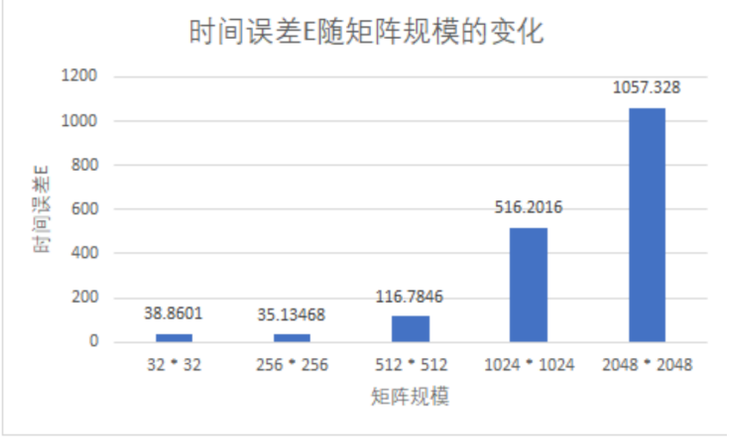
利用高速缓存改善程序效率的方式

通过对高速缓存性能以及相关运作机制的研究，可以得知在编写程序时需要充分利用缓存来提升代码的效率，尽量地去提高缓存的命中率。由于程序主要的时间都花在循环当中，所以优化要集中在优化核心函数的循环代码当中。而优化主要可以从两方面思考：时间局部性，应该在程序当中提高对局部变量的利用率，因为编译器能够将它们缓存在寄存器文件当中，利用时间局部性提高缓存的命中率；空间局部性，由于在上文提高的存储器层次结构当中所有层次上的缓存都是将数据存储为连续的块，所以应当尽量地避免步长过大地对不同块的数据进行访问。

下面用矩阵乘法的访存优化为例，分析在其中利用时间局部性以及空间局部性来改善程序效率的方式

```
for(int i = 0; i < row; i++){
    for(int k = 0; k < row; k++){
        float tmp = mat1 -> nums[i * col + k];
        for(int j = 0; j < col; j++){
            res -> nums[i * col + j] += tmp * mat2 -> nums[k * col + j];
        }
    }
}
```

通过将矩阵`mat1`的对应元素提前取出，利用时间局部性对该局部变量进行反复使用，避免重复地获取矩阵`mat1`对应元素，同时调换循环顺序，从行优先转化成为列优先，利用空间的局部性提高缓存的命中率。通过实际的实验测量，得到下面的实验结果图



可以发现随着矩阵规模的增大，朴素实现（未使用局部变量以及使用行优先顺序进行运算）与利用时间空间局部性提高缓存命中率的实现的时间误差$E = \frac{T_{long} - T_{short}}{T_{short}} * 100\%$会随着矩阵规模的增大而呈现指数变化的关系，从这个实验当中能够看出利用时空局部性对改善程序效率的显著作用。

同时，提升对时间局部性的利用程度还可以通过分块的策略实现，这样构造程序可以使得将一个程序块加载到L1的高速缓存当中，并在这个块当中进行所需的所有读写的操作，之后再丢弃这个块，加载下一个程序块，以此类推，这样也可以通过提升对时间局部性的利用来提升缓存命中。