

L1 cache —> L2 cache —> RAM

容量递增

速度递减

CPU访问优先级递减：先访问硬件cache，没有命中时访问RAM。

L1 指令和数据分别32K，cache line为64Bytes，因此有512个cache line。

硬件cache对应到内存的位置不是任意的：

cache line0 对应到内存地址：0~63，32K~32K+63，…

cache line1 对应到内存地址：64~127，32K+64~32K+127，…

cache line2 对应到内存地址：128~191，32K+128~32K+191，…

…

cache line512 对应到内存地址：32704~32767，32K+32703~32K+32767，…

因此，在定义数据类型、数据处理等尽量按照64字节对齐，提高命中率。

以下是一些简单测试情况：

#define ARR\_LEN 5\*1024

#define ONE\_ARR\_LEN 25\*1024\*1024

uint32\_t arr[ARR\_LEN][ARR\_LEN] = {0}; //size = 104857600

uint32\_t oarr[ONE\_ARR\_LEN] = {0}; //size = 104857600

int main(int argc, char \*argv[])

{

struct timeval t1, t2;

struct timezone tz;

uint32\_t time1, time2;

uint32\_t i, j;

//1 耗时=110 ms

for(i=0; i<ONE\_ARR\_LEN; i++)

{

oarr[i] \*= 3;

}

//2耗时=121 ms

for(i=0; i<ARR\_LEN; i++)

{

for(j=0; j<ARR\_LEN; j++)

arr[i][j] \*= 3;

}

//3耗时=61 ms

for(i=0; i<ARR\_LEN; i++)

{

for(j=0; j<ARR\_LEN; j++)

arr[i][j] \*= 3;

}

//4耗时= 1318 ms

for(i=0; i<ARR\_LEN; i++)

{

for(j=0; j<ARR\_LEN; j++)

arr[j][i] \*= 3;

}

}

上面程序静态编译，开启O2优化。

1和2测试，一维数组和二维数组顺序访问耗时差不多。

2和3测试，相同数据访问，3耗时减少一半，这是因为经过2测试数据已经一级或二级的cache中，3测试就快很多。这点提示：产生数据后尽量马上使用掉，提高命中，减少cache换入换出。

2和4测试，i和j的位置交换导致4为非顺序访问，cache line命中率低要经常切入和换出数据，导致耗时大大增加。这点提示：尽量顺序访问数据。