Cache Interference

Chapter 8

Origin of Cahce Interference

Thread local variables:

Each thread has an IntegerForLoop object. That object has three variables:

```
byte[] trialkey;        // new byte[32];
byte[] trialciphertext; // new byte[16];
AES256Cipher cipher;
```

JVM memory allocation

JVM usually allocates memory for variables one after another.

Local variables: For these three variables, three local four byte pointer variables are contructed: trielkey, trialciphertext, cipher.

Object and arrays: Objects and arrays are contructed in another section of the memory.

- 32 byte memory is allocated for trielkey,
- 16 bytes is allocated for trialciphertext,
- some number of bytes is allocated to cipher.

Memory layout

After each thread is constructed, memory layout could be as shown on Fig 9.2.

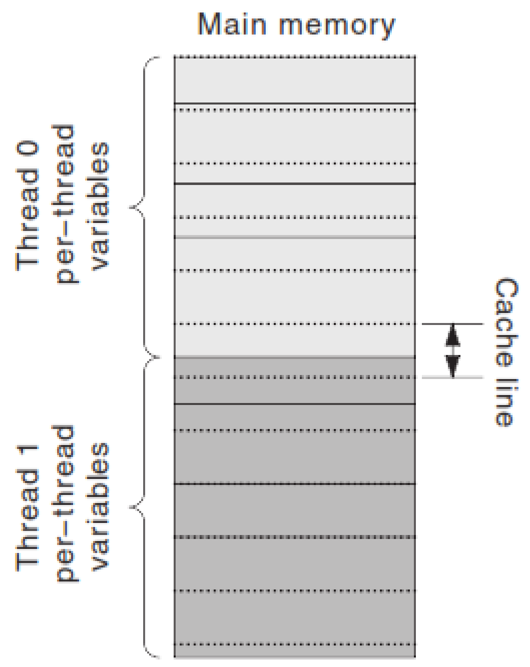

Figure 9.2 Memory layout showing cache line boundaries

Cache Interference

Cache Line size: A cache line is usually 64bytes or 128 bytes.

Variables from different threads: One cahce line may have variables from two or more threads. Although each variable belongs to different threads, they can be assigned on the same cache line.

Line Invalidation: If one thread updates its variable on a cache line, all the variables belonging to other threads on that cache line becomes invalid for those threads.

Reading from memory: Other threads read the new values from the memory again.

False Sharing: this is called false sharing. Since threads actually do not share any variables.

JVM fix: Actually JVM should fix this problem and should not assign variables from different threads to the same cache lines on the memory.

Parallel Java Fix

No cache line sharing: we must ensure that different threads’ per-thread variables never reside in the same cache line.

No memory management in Java: Java does not allow programmers to construct variables in different regions of the memory.

Padding: put some extra variables to each thread. Each thread occupies at least one more cache line.

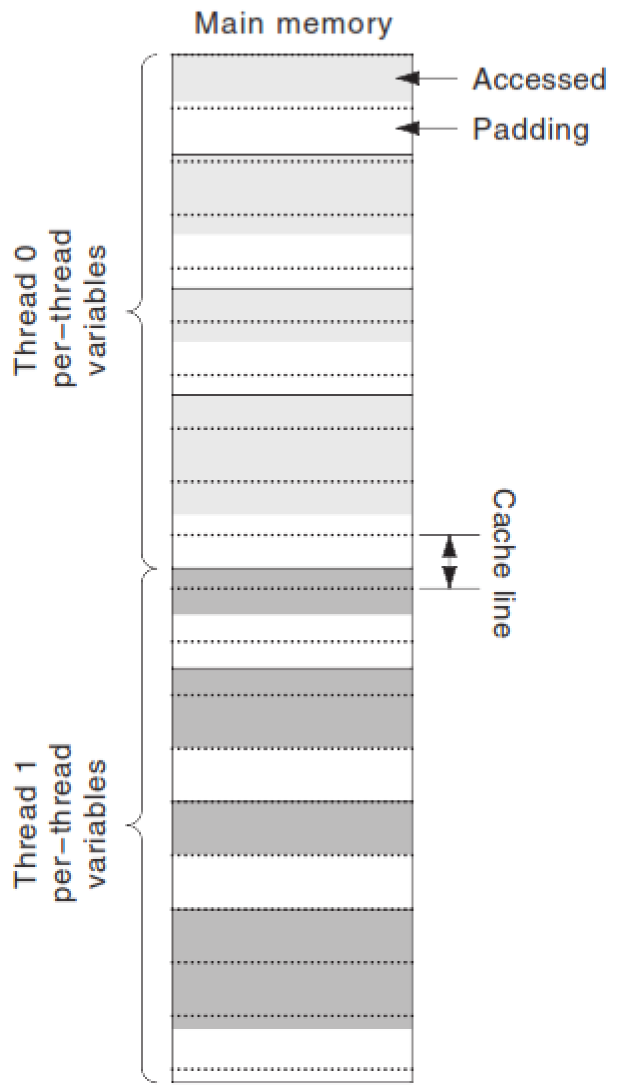

Figure 9.3 Memory layout with extra padding

Getting padding for local variables:

16 long variables occupy 16*8 = 128 bytes in memory.

It can not be a long array with 16 elements.

```
byte[] trialkey;         // new byte[32];
byte[] trialciphertext;  // new byte[16];
AES256Cipher cipher;
// Extra padding.
```

```
    long p0, p1, p2, p3, p4, p5, p6, p7;
    long p8, p9, pa, pb, pc, pd, pe, pf;
```

Getting padding for arrays

We make the array sizes at least 128 bytes.

```
    trialkey = new byte [32+128]; // + padding
    trialciphertext = new byte [16+128]; // + padding
```

Not a perfect solution: padding increases the amount of storage the program consumes.

Portabality reduced or extra memory consumed: It requires either knowing the machine’s cache line size (which reduces the program’s portability) or picking an amount of padding one hopes is larger than any machine’s cache line size (which may increase storage usage unnecessarily).

JVM Should fix it: But most of all, it requires that the programmer add code to do something the Java compiler or JVM should do. Perhaps a future version of the Java platform will automatically place per-thread variables in memory so as to avoid false sharing.

New Running Times from the book

Running times are almost linearly decreasing.

Almost constant efficiency.

Almost linear speedup.

Much better EDSF.

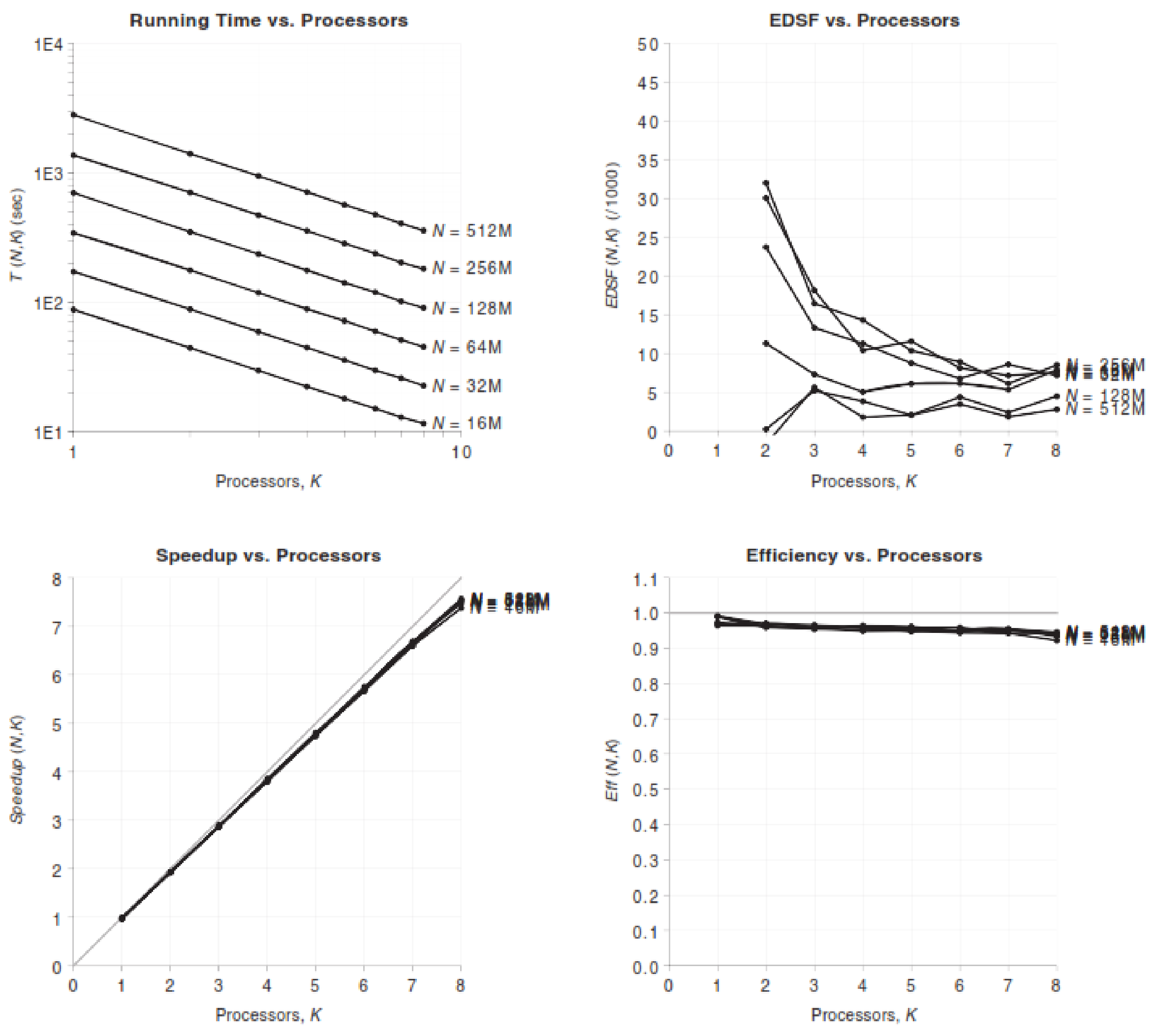

Figure 9.4 FindKeySeq/FindKeySmp3 running-time metrics