實驗八 Thread Safe & False Sharing

9617145 資工 4C 許晏峻
9617167 資工 4C 蔡孟儒

1. 實驗目的

利用 VTune 分析程式，了解範例程式中造成 false sharing 的地方，並且改善之。

2. 步驟過程

2.1. 基本題：thread-unsafe

由於 stock_sim 函式中的 seed 宣告為 static int 型態，代表著不同 thread 進來這個函式會共同使用這個變數，再加上函式也會對這個變數做寫入；因此，會造成 thread 在不同時間點使用這個變數時，值將無法預測，隱藏著 thread-unsafe 的問題。

所以這裡將 seed 宣告成 int *型態，並使用 malloc 動態分配一空間：

```
int *seed = malloc(sizeof(int));
*seed = 123;
```

函式 return 前再將這塊空間釋放掉：

```
free(seed);
```

2.2. 基本題：改善資料結構

由於 struct stock_t 資料結構的大小為 20 bytes，而每次 write data 後會做 cache coherence 的動作，將以 cache line 為單位做一致性的更新；然而，雖然其他 thread 使用的 data 在同一塊 memory 中，但之前寫入的 data 並不影響到需要使用的 data，卻因此 cache 被一起更新，造更過多的 cache miss(false sharing)。

所以這裡我們將 struct stock_t 資料結構的大小擴大為 64 bytes：

```
struct stock{
    unsigned int company_a;
    unsigned int company_b;
    unsigned int company_c;
    unsigned int index;
    unsigned int seed;
    int nop[11];
};
```

以上 int nop[11]就是為了將原本的 20 bytes 增加 44 bytes 變成 64bytes(cache

line size)，來避免 flase sharing problem。

2.3. 進階題：改善程式流程

由於 analyzer 函式中，for 迴圈使用 interleave 方式分工給各個 thread 完成任務，而這樣的流程導致 thread 會使用到同一塊 memory 並放到 cache 中，才會有 false sharing problem。這裡我們將改善 for 迴圈分工方式，改成以 block 為單位，避免不同 thread 使用同一塊 memory 造成的 false sharing(理論上只有邊界值可能發生)：

程式邏輯上，我們將每個 thread 的 for 迴圈的起始值(j)設為 stock_num/thread_num*threaded，邊界值為 start_stock+stock_num/thread_num；如此，每個 thread 使用的 data 的 memory address 將會是連續的位置，因此減少 false sharing 的機會。

3. 數據結果

實驗測試環境為：

- 實體機器 boot on USB 隨身硬碟
- CPU：Intel(R) Core(TM) i7 870
- Core：8
- Memory：4G
- OS：Ubuntu 10.04.01 TLS x86_64

	Origin (thread-safe)	基本題	進階題
Execution time	6.213s	5.641s	4.938s
L2 cache access count	1,692,400,000	1,823,200,000	449,200,000
L2 cache miss count	1,534,000,000	1,651,200,000	400,400,000
L1 cache miss rate	73.9761 %	65.4354 %	32.1705 %
L1 cache invalid count	12,000,000	4,000,000	4,000,000

表格 1 Vtune 檢測各項目數據

以下為截圖：

Elapsed Time: 6.213s

Paused Time: 0s

Hardware Events

Hardware Event Type	Hardware Event Count
CPU_CLK_UNHALTED.THREAD	152,770,000,000
L1D.M_SNOOP_EVICT	12,000,000
L1D_PREFETCH.MISS	346,800,000
L1D_PREFETCH.REQUESTS	468,800,000
L2_DATA_RQSTS.ANY	1,692,400,000
L2_RQSTS.LD_HIT	127,600,000
L2_RQSTS.LD_MISS	423,600,000
L2_RQSTS.MISS	1,534,000,000
MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM	68,640,000

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Command Line:	/home/ychsu/MCP/lab8/question/main.init 8 128 1000000
Frequency:	2.933 GHz
Logical CPU Count:	8
Operating System:	Linux
Computer Name:	ychsu-desktop
Result Size:	8 MB

圖 1 Origin 版本結果(僅修改為 thread-safe)

Elapsed Time: 5.641s

Paused Time: 0s

Hardware Events

Hardware Event Type	Hardware Event Count
CPU_CLK_UNHALTED.THREAD	139,548,000,000
L1D.M_SNOOP_EVICT	4,000,000
L1D_PREFETCH.MISS	297,600,000
L1D_PREFETCH.REQUESTS	454,800,000
L2_DATA_RQSTS.ANY	1,823,200,000
L2_RQSTS.LD_HIT	121,600,000
L2_RQSTS.LD_MISS	313,600,000
L2_RQSTS.MISS	1,651,200,000

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Command Line:	/home/ychsu/MCP/lab8/question/main.64 8 128 1000000
Frequency:	2.933 GHz
Logical CPU Count:	8
Operating System:	Linux
Computer Name:	ychsu-desktop
Result Size:	7 MB

圖 2 基礎題版本結果

Elapsed Time: 4.938s

Paused Time: 0s

Hardware Events

Hardware Event Type	Hardware Event Count
CPU_CLK_UNHALTED.THREAD	121,592,000,000
L1D.M_SNOOP_EVICT	4,000,000
L1D_PREFETCH.MISS	66,400,000
L1D_PREFETCH.REQUESTS	206,400,000
L2_DATA_RQSTS.ANY	449,200,000
L2_RQSTS.LD_HIT	24,000,000
L2_RQSTS.LD_MISS	87,600,000
L2_RQSTS.MISS	400,400,000
MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM	5,200,000

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Command Line:	/home/ychsu/MCP/lab8/question/main.block 8 128 1000000
Frequency:	2.933 GHz
Logical CPU Count:	8
Operating System:	Linux
Computer Name:	ychsu-desktop
Result Size:	6 MB

圖 3 進階題版本結果

4. 結論心得

從這次實驗中，一開始我們將 thread-unsafe 的程式改寫成 thread-safe 程式，印證了上課學到的概念，除了速度上有很大的差異外(因為 thread 都共用到同一個 static 變數)，這樣做法才是較正確的程式邏輯。

之後我們學習到 thread 之間還有 false sharing 的問題，這將會大大地影響到程式的效能；而解決辦法基本上有：從資料結構改善和從程式流程改善兩種方向。而這兩種方式我們都在實驗中嘗試完成，並且確實可以明顯看到效能上有很大的改變，這些問題實在都是平常不會特別想到的問題，很有意思！