HW14 thread

408410019 資工二 徐怡娟

1. 程式可以用蒙特卡羅法算出 Pi

```
pi = 3.14165699

real    0m6.879s
user    0m27.279s
sys     0m0.040s
```

2. 按下 ctrl-c 可以顯示截至目前算出來的 Pi

```
^Chit: 130494973, loop: 166149830, pi = 3.14162158
```

3. 一秒內連續按下二次 ctrl-c 顯示截至目前算出的 Pi 並結束程式

```
^Chit: 130494973, loop: 166149830, pi = 3.14162158
^Chit: 552156823, loop: 703025976, pi = 3.14160126
time used: 2.237322
^Chit: 749559762, loop: 954354234, pi = 3.14164169
time used: 1.059097
^Chit: 902155592, loop: 1148636261, pi = 3.14165806
time used: 0.807655

real    0m4.796s
user    0m18.870s
sys     0m0.036s
```

4. 與範例程式比較

範例程式

total_loop/thread	1600000000/16	1600000000/8	1600000000/4
real time	19.279s	26.645s	26.889s
user time	76.143s	105.560s	105.594s
sys time	0.164s	0.136s	0.092s

自己的程式(mutex+aligned)

total_loop/thread	1600000000/16	1600000000/8	1600000000/4
real time	6.852	7.046	7.157
user time	27.181	27.852	28.276
sys time	0.036	0.030	0.036

自己的程式(atomic operation+aligned)

total_loop/thread	1600000000/16	1600000000/8	1600000000/4
real time	6.869	7.112	7.399
user time	27.236	28.080	29.237
sys time	0.034	0.049	0.029

(以上表格結果為五次執行結果平均四捨五入取到小數點後第三位)

我對此程式進行了二個優化，因此能比範例程式快

(1) 將每個 thread 需要個別儲存的資料包成一個 struct，一進入 thread()時就在參數傳入各自的 struct，省去進入 thread()時用 mutex 來避免 idx 同時被存取而出錯

使用

```
typedef struct th_pack {    // sizeof 40 bytes
    long volatile hit;  // sizeof 8 bytes
    long volatile loopcount;    // sizeof 8 bytes
    struct drand48_data rand_buf;   // sizeof 24 bytes
} Thread_Pack;
```

```
Thread_Pack* th_data[16];
```

```
pthread_create(&id[i],NULL,(void *)thread, th_data[i]);
```

避免掉（mutex 底層是利用 atomic operation 實現，overhead 較高）

```
pthread_mutex_lock(&mutex);
volatile long* local_hit = &hit[idx];
volatile long* cur_loopcount = &loopcount[idx++];
pthread_mutex_unlock(&mutex);
```

(2) 整個 struct 的 size 為 40bytes，再利用 aligned_alloc 強制讓記憶體對齊，速度表現比沒使用好很多，推測是因為若未使用 aligned_alloc，雖然 struct size 為 8bytes 的整數倍但若記憶體開頭未被放在 8bytes 整數倍的位置，仍會多幾次 CPU 讀取次數

```
th_data[i] = aligned_alloc(40 , sizeof(struct th_pack));
```

1600000000/16	使用 aligned_alloc	未用 aligned_alloc
real time	6.852	11.017
user time	27.181	42.478
sys time	0.036	0.024

(表格結果為五次執行結果平均四捨五入取到小數點後第三位，mutex 版本)

嘗試將範例程式中第二個 mutex 拿掉換成 atomic_fetch_add（atomic operation）來將每個 thread 的 local_hit 合併到 global_hit 上，在 10^9 ~10^{10} 等級的總 loop 數，兩者速度表現差不多，原本預測當數量級拉高時差距會非常大，推測是這段只有在最後加總使用 mutex、overhead 沒有那麼高，再加上 critical section 很短造成速度差距沒有預測懸殊。

使用

```
atomic_long global_hit = 0;
```

```
atomic_fetch_add(&global_hit, pack->hit);
```

代替

```
pthread_mutex_lock(&mutex);
global_hit += pack->hit;
pthread_mutex_unlock(&mutex);
```

1600000000/4	atomic_operation	mutex
real time	7.399	7.157
user time	29.238	28.276
sys time	0.029	0.036

(表格結果為五次執行結果平均四捨五入取到小數點後第三位)

10000000000/4	atomic_operation	mutex
real time	51.480	52.378
user time	202.480	205.191
sys time	0.142	0.117

(表格結果為五次執行結果平均四捨五入取到小數點後第三位)