軟體分析與最佳化 期中

612410017 林靖紳

Evaluation configuration

- CPU information

```
ashen@Stephanie-Lin:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz
    CPU family:          6
    Model:               167
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            1
    CPU max MHz:         4600.0000
    CPU min MHz:         800.0000
    BogoMIPS:            5424.00
```

- Memory

```
ashen@Stephanie-Lin:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            31Gi       4.2Gi        12Gi       1.8Gi        14Gi        24Gi
Swap:          2.0Gi          0B       2.0Gi
```

- OS version

```
ashen@Stephanie-Lin:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.2 LTS
Release:        22.04
Codename:       jammy
```

- GCC version

```
ashen@Stephanie-Lin:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- ICC version

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW2$ icc --version
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from
product release in the second half of 2023. The Intel(R) oneAPI DPC++/C++ Compiler (ICX) is the rec
ommended compiler moving forward. Please transition to use this compiler. Use '-diag-disable=10441'
 to disable this message.
icc (ICC) 2021.10.0 20230609
Copyright (C) 1985-2023 Intel Corporation.  All rights reserved.
```

2. gzip

(1) gcc -O0

- Change compile option

```
415  CC = gcc
416  CCDEPMODE = depmode=gcc3
417  CFLAGS = -g -O0
418  CONFIG_INCLUDE = lib/config.h
419  CPP = gcc -E
420  CPPFLAGS =
421  CYGPATH_W = echo
422  DEFS = -DHAVE_CONFIG_H -DUNIX
```

- Compile and execute gprof

```
asheh@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid/gzip/gzip-1.10$ gprof ./gzip > ../../test_O0_result.txt
```

- The top five functions that have most CPU time

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 44.68      0.21     0.21        1   210.00   469.88  deflate
 19.15      0.30     0.09      512     0.18     0.23  fill_window
 17.02      0.38     0.08 10546671     0.00     0.00  longest_match
 10.64      0.43     0.05      512     0.10     0.10  copy_block
  6.38      0.46     0.03      513     0.06     0.06  updcrc
```

(2) gcc -O3

- Compile option

```
415  CC = gcc
416  CCDEPMODE = depmode=gcc3
417  CFLAGS = -g -O3 -pg
418  CONFIG_INCLUDE = lib/config.h
419  CPP = gcc -E
420  CPPFLAGS =
421  CYGPATH_W = echo
422  DEFS = -DHAVE_CONFIG_H -DUNIX
```

- The top five

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 68.18      0.15     0.15        1   150.00   219.96  deflate
 13.64      0.18     0.03 16768796     0.00     0.00  ct_tally
  9.09      0.20     0.02 10546671     0.00     0.00  longest_match
  4.55      0.21     0.01   274868     0.00     0.00  pqdownheap.constprop.0
  4.55      0.22     0.01      512     0.02     0.02  copy_block
```

(3) What functions are directly called by the function deflate()? How many times are they each called by the function deflate()?

- It's the gprof result with the compile option "-O3"

```
100 -----------------------------------------------
101                0.15    0.07       1/1           zip [2]
102 [4]    100.0    0.15    0.07       1          deflate [4]
103                0.03    0.00 16768796/16768796     ct_tally [5]
104                0.02    0.00 10546671/10546671     longest_match [6]
105                0.00    0.02     511/512         flush_block [7]
106                0.00    0.00     512/512         fill_window [18]
107 -----------------------------------------------
```

functions	called times
ct_tally	16768796
longest_match	10546671
flush_block	511
fill_window	512

(4) icc -O3

- compile option

```
415  CC = icc
416  CCDEPMODE = depmode=gcc3
417  CFLAGS = -g -O3 -pg
418  CONFIG_INCLUDE = lib/config.h
419  CPP = gcc -E
420  CPPFLAGS =
421  CYGPATH_W = echo
422  DEFS = -DHAVE_CONFIG_H -DUNIX
```

- The top five functions that have most CPU time

```
1 Flat profile:
2
3 Each sample counts as 0.01 seconds.
4   %   cumulative   self              self     total
5  time   seconds   seconds    calls  ms/call  ms/call  name
6  31.58      0.06     0.06        1    60.00   189.84  deflate
7  21.05      0.10     0.04      513     0.08     0.08  updcrc
8  21.05      0.14     0.04      512     0.08     0.08  copy_block
9  15.79      0.17     0.03 16768796     0.00     0.00  ct_tally
10  5.26      0.18     0.01 10546671     0.00     0.00  longest_match
11  5.26      0.19     0.01     1536     0.01     0.01  build_tree
12  0.00      0.19     0.00   139288     0.00     0.00  bi_reverse
13  0.00      0.19     0.00     1025     0.00     0.00  flush_outbuf
14  0.00      0.19     0.00      513     0.00     0.08  file_read
15  0.00      0.19     0.00      513     0.00     0.00  read_buffer
16  0.00      0.19     0.00      512     0.00     0.10  flush_block
17  0.00      0.19     0.00      512     0.00     0.00  send_bits
```

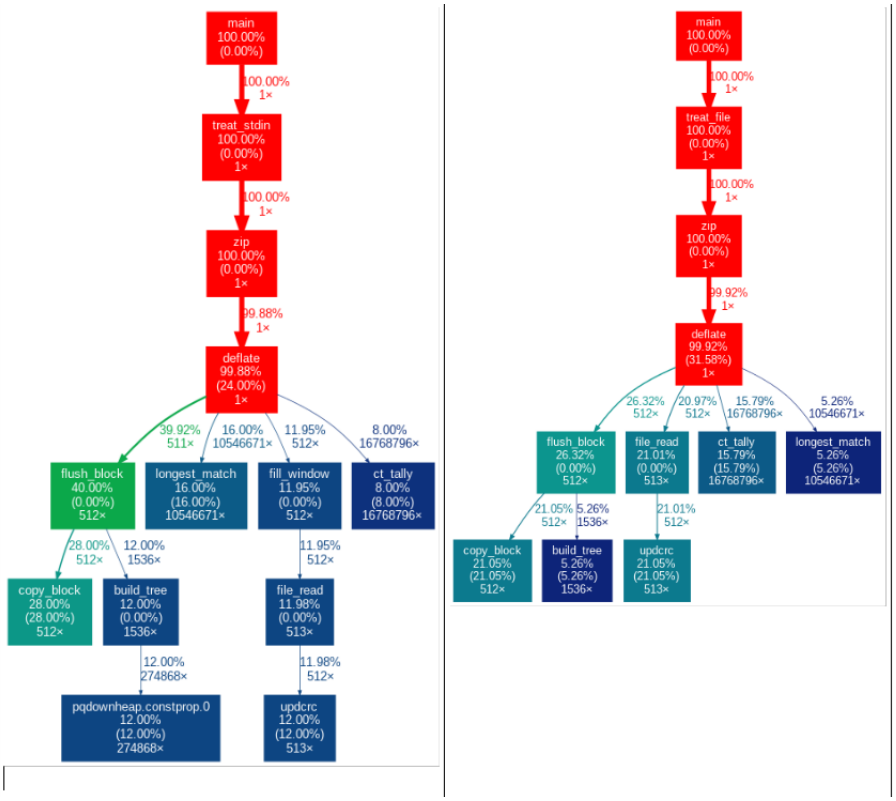
- Compare the results with (2)

- 在 gcc 編譯下，主要時間花費在 copy_block、deflate 和 longest_match 這些函數上。
  - copy_block 佔用了 28% 的總執行時間，並且它的自身執行時間很高。
  - deflate 佔用了 24% 的總執行時間，但它的自身執行時間相對較低。
  - longest_match 佔用了 16% 的總執行時間。
  - 另外，ct_tally 函數佔用了 8% 的時間。

- 在 icc 編譯下，主要時間花費在 deflate、updcrc 和

copy_block 函數上。

- deflate 佔用了 31.58% 的總執行時間，並且它的自身執行時間較高。
- updcrc 佔用了 21.05% 的總執行時間，和 copy_block 也佔用了 21.05%。
- ct_tally 佔用了 15.79% 的時間。
- longest_match 佔用了 5.26% 的時間。
- 其他函數的佔用時間相對較低，但和 deflate、updcrc 和 copy_block 有關。



3. Codecov PrimarySingle.cpp

- Compile

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ icc PrimeSingle.cpp -O3 -o pris.exe -lm
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in t
ed compiler moving forward. Please transition to use this compiler. Use '-diag-disable=10441' to disable this message.
```

- Execute

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ ./pris.exe 1 10000000
100%

  664579 primes found between      1 and 10000000 in    1.35 secs
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$
```

- generate the optimization reports

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ icc PrimeSingle.cpp -O3 -o pris.exe -lm -qopt-report
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in t
ed compiler moving forward. Please transition to use this compiler. Use '-diag-disable=10441' to disable this message.
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$
```

(1) Is the loop at line 100 vectorized by the compiler ?

- yes!

```
35 LOOP BEGIN at PrimeSingle.cpp(111,5) inlined into PrimeSingle.cpp(131,5)
36    remark #15542: loop was not vectorized: inner loop was already vectorized
37
38    LOOP BEGIN at PrimeSingle.cpp(100,5) inlined into PrimeSingle.cpp(131,5)
39       remark #15300: LOOP WAS VECTORIZED
40    LOOP END
41
42    LOOP BEGIN at PrimeSingle.cpp(100,5) inlined into PrimeSingle.cpp(131,5)
43    <Remainder loop for vectorization>
44    LOOP END
45 LOOP END
```

(2) Whether or not the function TestForPrime() is inlined?

- TestForPrime is inlined

```
219 INLINE REPORT: (FindPrimes(int, int)) [6] PrimeSingle.cpp(107,1)
220   -> INLINE: (113,13) TestForPrime(int)
221   -> INLINE: (116,9) ShowProgress(int, int)
222
```

(3) Please use Intel codecov to do code coverage analysis and answer the following questions.What are the code coverages? (from the point of view of Functions and Blocks) What are the execution counts for lines 103 and 114?

- Compile
  - generate *.spi

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ icc -DUNIX -O3 -prof-gen=srcpos PrimeSingle.cpp -o PrimeSingle_icc
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in the second hal
  '-diag-disable=10441' to disable this message.
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ ls
gzip  midterm_2023fall_v1.pdf  pgopti.spl  pgopti.spl  PrimeSingle.cpp  PrimeSingle_icc  PrimeSingle.optrpt  pris.exe
```

  - Execute to generate *.dyn

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ ./PrimeSingle_icc 1 10000000
100%

  664579 primes found between      1 and 10000000 in   23.07 secs
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$
```

  - Execute “profmerge” to generate *.dpi

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ profmerge
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ ls -al
total 356
drwxrwxr-x 3 ashen ashen   4096 ┼─   7 11:51 .
drwxrwxr-x 8 ashen ashen   4096 ┼─   7 10:16 ..
-rw-rw-r-- 1 ashen ashen   1144 ┼─   7 11:48 6549b3ab_28841.dyn
-rw-rw-r-- 1 ashen ashen   3088 ┼─   7 11:49 6549b3d6_28846.dyn
drwxrwxr-x 3 ashen ashen   4096 ┼─   7 11:38 gzip
-rw-rw-r-- 1 ashen ashen 162473 ┼─   7 10:15 midterm_2023fall_v1.pdf
-rw-rw-r-- 1 ashen ashen   5992 ┼─   7 11:51 pgopti.dpi
```

- Use codecov

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ codecov -counts -prj PrimeSingle_icc -spi pgopti.spl -dpi pgopti.dpi -txtbcvrg result.txt
Intel(R) C++/Fortran Compiler Classic code-coverage tool, Version 2021.10.0 Build 20230609_000000
Copyright (C) 1985-2023 Intel Corporation.  All rights reserved.

ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$
```

- Summary

Files				Functions				Blocks			
total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%
2	2	0	**100.00**	6	6	0	**100.00**	45	37	8	**82.22**

Covered Files in PrimeSingle_icc

Name	Functions			Blocks		
	total	cvrd	cvrg%	total	cvrd	cvrg%
PrimeSingle.cpp	5	5	100.00	40	35	87.50
stdlib.h	1	1	100.00	5	2	40.00

- **What are the code coverages?**
  - Functions
    - 追蹤程式碼中的每個函數是否在測試中至少執行一次
    - 確保所有程式中的函數都經過測試，有助於識別未測試或未使用的函數。
  - Blocks
    - 測量在測試期間至少執行過一次的程式碼區塊（通常表示為程式碼行數）的數量。
    - 區塊是函數內的程式碼部分，可以是單行或多行代碼

- Execution counts for lines 103： 4,999,999

```
103)     return (factor > limit);
                 ^ 4,999,999
104) }
```

- Execution counts for lines 114: 664,578

```
113)         if( TestForPrime(i) )
                 ^ 9,999,998 (2)
114)             globalPrimes[gPrimesFound++] = i;
                 ^ 664,578
```

4. Vtune PrimeSingle.cpp

- Compile

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ icc -g -o pris.exe -lm
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be re
ed compiler moving forward. Please transition to use this compiler. Use '-diag-disable=1
icc: warning #10315: specifying -lm before files may supersede the Intel(R) math library
icc: command line error: no files specified; for help type "icc -help"
```

- Execute

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$ ./pris.exe 1 4000000
100%

  283146 primes found between      1 and 4000000 in    0.39 secs
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/Mid$
```

(a) Run “Performance Snapshot”. What is its elapsed time?

- Elapsed time: 0.681

Elapsed Time: 0.681s

IPC:	2.620
SP GFLOPS:	0.015
DP GFLOPS:	0.000
x87 GFLOPS:	0.000
Average CPU Frequency:	4.5 GHz

Logical Core Utilization: 8.3% (0.992 out of 12)

Microarchitecture Usage: 56.4% of Pipeline Slots

Memory Bound: 4.8% of Pipeline Slots

Vectorization: 0.0% of Packed FP Operations

(b) Software-mode sampling" to do Hotspots analysis

- Elapsed time: 0.890

Elapsed Time: 0.890s

CPU Time:	0.660s
Total Thread Count:	1
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
TestForPrime	pris.exe	0.624s	94.5%
__libm_sqrtf_e7	pris.exe	0.024s	3.6%
ShowProgress	pris.exe	0.012s	1.8%

*N/A is applied to non-summable metrics.

- Which line has the most CPU time?
  - The function TestForPrime had the most CPU time
  - It’s defined in from line 95 to 103
    - The while loop in line 100 took the most CPU time.

(c) According to the analysis results, please give some ideas to improve its performance

- 由於程式的 Hotspot 在 TestForPrime 的函式中
- 這個函數的目標是測試一個整數 val 是否為質數，它使用了一個簡單的 while 迴圈來檢查是否存在能整除 val 的因子
- 可以跳過已知的非質數：
  - 可以建立一個小質數表，用它們來先測試 val 是否可以整除。這就可以減少因子測試的次數。

5. There are two methods used to collect performance data: instrumentation and sampling.

(1) What is instrumentation

- 目的:
  - 主要目的是深入了解程序或系統在運行時的行為，識別效能瓶頸、記憶體洩漏或其他可能影響效能和可靠性的問題。
- 方法:
  - 插入測試程式到目標的程式/系統中，以在執行期間監視和記錄數據。
- 優點:
  - 可提供高度細粒度的數據
  - 通過直接測量和記錄程式中的數據，收集特定和準確的效能指標。
- 缺點:
  - 但是會增加執行時間開銷。這些附加的程式碼需要額外的處理能力和記憶體，這可能對軟體的效能產生一定程度的負擔。
  - 監控和記錄應用程式的內部行為可能會暴露敏感數據或漏洞。

(2) Please explain how hardware event-based sampling works

- 目的:
  - Hardware Event-Based Sampling 用於收集計算機系統性能數據的方法
- 方法:
  - Performance Counters
    - 硬體性能計數器，這些計數器是特殊寄存器，用於計算特定的硬體事件。這些事件可以包括快取未命中、分支錯誤預測、CPU週期等等
  - Event Selection
    - 進行硬體事件取樣，選擇一個或多個特定的硬體事件進行監控
  - Sampling Period

        - 指定取樣週期或間隔，以確定硬體計數器何時進行取樣。
    - Sampling Mechanism
        - 每個取樣週期結束時，硬體自動觸發中斷，並保存CPU的狀態
    - Data Collection
        - 收集的數據通常存儲在專用緩衝區或內存區域中
    - Analysis
        - 分析以上收集的數據

6. Consider the following loop. Does it exist a loop-carried/loop-independent data dependence between S1 and S2? Explain your answer

```
int m;
m = func();
for (i=1; i<20; i++) {
    A[i+m] = B[i] + 2;
    C[i] = A[i] - 1;
}
```

- 如果 0<m<20 ，有 loop-carried dependencies
    - 因為 S1 的 A[i+m] 在 m 範圍介在 1 到 19 之間的時候，一定會跟 S2 的 A[i] 產生跨 iteration 的迭代關係
- 如果 m =0, S1 和 S2 有 loop-independent 的關係
    - 當 m == 0 時，S1 的 A[i+0] 和 S2 的 A[i] 直接會有 loop independent 的關係