軟體分析與最佳化 HW4

612410017 林靖紳

編譯

1. 編譯

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW4$ icc nsieve.c -O2 -DUNIX -qopt-report -o nsieve
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product
 '-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/HW4$
```

2. qopt-report

```
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.


    Report from: Interprocedural optimizations [ipo]

INLINING OPTION VALUES:
  -inline-factor: 100
  -inline-min-size: 30
  -inline-max-size: 230
  -inline-max-total-size: 2000
  -inline-max-per-routine: 10000
  -inline-max-per-compile: 500000


Begin optimization report for: main()

    Report from: Interprocedural optimizations [ipo]

INLINE REPORT: (main()) [1] nsieve.c(152,1)
  -> (189,1) SIEVE()
  -> (222,1) SIEVE()
  -> (235,4) SIEVE()


    Report from: Loop nest, Vector & Auto-parallelization optimizations [loop, vec, par]

```

問答

Q1: LINE 329所在的LOOP CONSTRUCT是否有被編譯器向量化? (HINT:“-QOPT-REPORT” OPTION)

- 是，line 329 所在的迴圈有被編譯器向量化
- 根據下圖，在第一次 iteration 時，有一個 remark #15542 顯示此迴圈並未被向量化

```
LOOP BEGIN at nsieve.c(325,4)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at nsieve.c(329,4)
      remark #25408: memset generated
      remark #15542: loop was not vectorized: inner loop was already vectorized

      LOOP BEGIN at nsieve.c(329,4)
         remark #15300: LOOP WAS VECTORIZED
      LOOP END

      LOOP BEGIN at nsieve.c(329,4)
      <Remainder loop for vectorization>
      LOOP END
   LOOP END

   LOOP BEGIN at nsieve.c(335,6)
      remark #15541: outer loop was not auto-vectorized: consider using SIMD directive

      LOOP BEGIN at nsieve.c(341,2)
         remark #15335: loop was not vectorized: vectorization possible but seems inefficient. U
override
         remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
      LOOP END
   LOOP END
LOOP END
```

- 而接著在第二次 iteration 的時候，有一個 remark #15300 顯示此迴圈已經被向量化了。

```
LOOP BEGIN at nsieve.c(325,4)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at nsieve.c(329,4)
      remark #25408: memset generated
      remark #15542: loop was not vectorized: inner loop was already vectorized

      LOOP BEGIN at nsieve.c(329,4)
         remark #15300: LOOP WAS VECTORIZED
      LOOP END

      LOOP BEGIN at nsieve.c(329,4)
      <Remainder loop for vectorization>
      LOOP END
   LOOP END

   LOOP BEGIN at nsieve.c(335,6)
      remark #15541: outer loop was not auto-vectorized: consider using SIMD directive

      LOOP BEGIN at nsieve.c(341,2)
         remark #15335: loop was not vectorized: vectorization possible but seems inefficient. U
override
         remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
      LOOP END
   LOOP END
LOOP END
```

Q2: GCC COMPILER提供了編譯參數"-FFAST-MATH"，請問它的主要功能是什麼? 適合的使用情境是什麼?

- GCC編譯器提供的 “-ffast-math” 參數的主要功能是優化一些數學運算，主要針對浮點數運算
- 功能如下
  - 移除一些浮點數運算的精確度和一致性要求，例如關於NaN（非數值）和無窮大的處理。這可以導致更快的數學運算，但可能會產生非 IEEE 標準的結果。
  - 允許進行循環展開和重排序浮點數運算，以提高性能。這可能會改變運算順序導致微小的精確度損失。
  - 啟用一些特定的編譯器優化，如合併浮點數運算，消除冗餘的計算等，以減少程式碼大小
- 適用情境：
  - 適用於那些不要求極高浮點精確度，更關注性能而不是精確度的應用。
  - 如圖形渲染、遊戲開發、機器學習和深度學習等等

以 nsieve.c 為例執行 gcc -ffast-math 之比較

- 以 gcc 編譯之執行結果

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW4$ gcc nsieve.c -O2 -DUNIX -o nsieve_gcc
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW4$ ./nsieve_gcc

   Sieve of Eratosthenes (Scaled to 10 Iterations)
   Version 1.2b, 26 Sep 1992

   Array Size    Number    Last Prime     Linear     RunTime     MIPS
    (Bytes)    of Primes                Time(sec)    (Sec)
       8191        1899         16381      0.000      0.000  17783.1
      10000        2261         19997      0.000      0.000  19899.2
      20000        4202         39989      0.000      0.000  11526.8
      40000        7836         79999      0.000      0.001  9967.9
      80000       14683        160001      0.001      0.002  8683.1
     160000       27607        319993      0.002      0.004  8446.7
     320000       52073        639997      0.004      0.008  8281.8
     640000       98609       1279997      0.007      0.019  7313.6
    1280000      187133       2559989      0.015      0.044  6323.7
    2560000      356243       5119997      0.029      0.091  6253.5
    5120000      679460      10239989      0.058      0.197  5817.9
   10240000     1299068      20479999      0.117      0.416  5545.4
   20480000     2488465      40960001      0.233      1.495  3107.8
   40960000     4774994      81919993      0.466      4.008  2336.3

   Relative to 10 Iterations and the 8191 Array Size:
   Average RunTime =    0.000 (sec)
   High  MIPS      =  19899.2
   Low   MIPS      =   2336.3
```

- 加上參數 -ffast-math

```
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW4$ gcc nsieve.c -O2 -DUNIX -ffast-math -o nsieve_exe
ashen@Stephanie-Lin:~/Documents/Software_Analysis-git/HW4$ ./nsieve_exe

   Sieve of Eratosthenes (Scaled to 10 Iterations)
   Version 1.2b, 26 Sep 1992

   Array Size    Number    Last Prime     Linear     RunTime     MIPS
    (Bytes)    of Primes                Time(sec)    (Sec)
       8191        1899         16381      0.000      0.000  21082.2
      10000        2261         19997      0.000      0.000  19087.7
      20000        4202         39989      0.000      0.000  11000.1
      40000        7836         79999      0.000      0.001  9507.0
      80000       14683        160001      0.001      0.002  8510.0
     160000       27607        319993      0.002      0.004  8278.9
     320000       52073        639997      0.003      0.009  8066.6
     640000       98609       1279997      0.006      0.019  7348.2
    1280000      187133       2559989      0.012      0.043  6519.3
    2560000      356243       5119997      0.025      0.085  6663.3
    5120000      679460      10239989      0.049      0.186  6161.6
   10240000     1299068      20479999      0.098      0.412  5592.1
   20480000     2488465      40960001      0.197      1.477  3147.0
   40960000     4774994      81919993      0.393      3.946  2372.9

   Relative to 10 Iterations and the 8191 Array Size:
   Average RunTime =    0.000 (sec)
   High  MIPS      =  21082.2
   Low   MIPS      =   2372.9
```

- 分析
  - 使用"-ffast-math"：
    - MIPS 值較高，表示程式在相同時間內執行了更多的指令。
    - 隨著數組大小的增加，MIPS值也隨之下降，但整體上仍然保持較高的性能。