ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №4

Выполнил:
Пятаев Егор, гр. 15206

Постановка задачи

Вариант задания: DC.

Для программы, выбранной в соответствии с вариантом задания, выполнить следующие действия:

1.Выполнить автоматическую оптимизацию с помощью компилятора, используя информацию от профилировщика. Оценить полученное ускорение.
2.Оценить накладные расходы на профилирование программы.
3.Построить граф вызовов программы (картинку). С помощью графа вызовов определить «горячие точки» программы с точностью до функций.
4.Построить аннотированный листинг программы. Определить «горячие точки» программы с точностью до строк исходного кода и команд ассемблера. Сделать предположение о том, что является основной причиной временных затрат (вычислительные операции, обращения в память, выполнение команд передачи управления, …).
5.Собирая информацию о соответствующих событиях, получить с помощью профилирования следующую информацию о «горячих» функциях и о программе в целом:

- среднее число тактов на микрооперацию,
- процент кэш-промахов от всех обращений для кэшей первого и последнего уровней,
- процент неправильно предсказанных переходов.

На основании полученной информации сделать вывод о причинах временных затрат при выполнении программы. Можно собрать информацию и о других событиях, если это требуется для обоснования вывода.

Результаты

1.
    1. -fprofile-generate

```
real    19m37,370s
user    9m45,364s
sys     0m29,644s
```

    2. -fprofile-use

```
real    18m23,761s
user    8m14,064s
sys     0m31,376s
```

2. -O1

```
real    17m40,859s
user    7m39,848s
sys     0m28,456s
```

3. Профиль: report.txt, cache.profile.txt
   Граф вызовов: callgraph.png

4. annotation.txt, source.profile.txt

Cache-misses:

```
       │    TreeInsert:
1,10   │      push   %r15
       │      push   %r14
       │      push   %r13
       │      push   %r12
0,51   │      push   %rbp
       │      push   %rbx
2,88   │      sub    $0x8,%rsp
```

```
       mov    %rdi,%rbp
       mov    %rsi,%r14
 2,95  mov    (%rdi),%r12
 3,32  test   %r12,%r12
     ↑ je     38
 1,70 ba:   mov    0x50(%rdi),%rax
 2,22       movl   $0x0,(%rax)
 1,27       mov    0x58(%rdi),%rdx
 1,69       mov    %rdi,(%rdx)
 1,81       mov    $0x2,%r13d
 4,10     ↑ jmpq   bf
 1,49       addl   $0x1,0x28(%rdi)
17,62       mov    0x38(%rdi),%r11d
 0,32       mov    %r11d,%r12d
       add    0x60(%rdi),%r12
 2,07       mov    %r12,0x20(%rdi)
 0,25       add    0x2c(%rdi),%r11d
 2,10       mov    %r11d,0x38(%rdi)
 6,61       mov    0x40(%rdi),%edi
          ….
```

KeyComp:

```
 0,81       add    %al,%bh
 0,40       sub    %al,(%rax)
 0,68       add    %al,(%rax)
42,87       add    %al,%bh
       cmp    %al,(%rax)
 1,97       add    %al,(%rax)
 1,44       add    %al,-0x73(%rbx)
       rex.WR
 0,09       gs     (bad)
 0,10       shl    $0x2,%ecx
       mov    %ecx,%eax
 4,77       and    $0x7,%eax
       lea    (%rcx,%rax,1),%edx
 3,00       mov    %edx,0x2c(%rbx)
 2,21       mov    %ebp,0x34(%rbx)
 6,27       movl   $0x0,0x3c(%rbx)
12,86       sub    $0x1c,%ecx
 4,31       mov    %ecx,0x30(%rbx)
 5,37       movq   $0x0,0x20(%rbx)
 0,01       mov    %ebp,%eax
 0,00       mov    $0x0,%edx
       divl   0x2c(%rbx)
 0,27       mov    %eax,0x44(%rbx)
       mov    %eax,0x40(%rbx)
 0,37       mov    %r13d,0x48(%rbx)
 0,08       mov    %r12d,0x4c(%rbx)
 7,93       mov    %r14,0x60(%rbx)
 4,16       .byte  0xbf
              ...
```

Branch-misses:

TreeInsert:

```
          mov    %eax,0x44(%rdi)
       mov    %eax,0x40(%rdi)
 0,50  mov    %esi,0x48(%rdi)
30,14 9a:   mov    %ecx,0x4c(%rdi)
```

```
       ←  retq

       000000000000594e <TreeInsert>:
       TreeInsert():
 0,01  │     push   %r15
       │     push   %r14
       │     push   %r13
       │     push   %r12
 0,07  │     push   %rbp
       │     push   %rbx
 0,05  │     sub    $0x8,%rsp
       │     mov    %rdi,%rbp
       │     mov    %rsi,%r14
 0,32  │     mov    (%rdi),%r12
 0,02  │     test   %r12,%r12
       │   ↑ je     38
 0,74  │ ba: mov    0x50(%rdi),%rax
 0,22  │     movl   $0x0,(%rax)
 0,01  │     mov    0x58(%rdi),%rdx
 0,60  │     mov    %rdi,(%rdx)
 0,11  │     mov    $0x2,%r13d
 0,18  │   ↑ jmpq   bf
 0,44  │     addl   $0x1,0x28(%rdi)
 4,75  │     mov    0x38(%rdi),%r11d
 0,05  │     mov    %r11d,%r12d
       │     add    0x60(%rdi),%r12
 0,58  │     mov    %r12,0x20(%rdi)
 0,00  │     add    0x2c(%rdi),%r11d
 4,11  │     mov    %r11d,0x38(%rdi)
28,72  │     mov    0x40(%rdi),%edi
 0,01  │     sub    $0x1,%edi
       │     mov    %edi,0x40(%rbp)
       │     test   %edi,%edi
 1,13  │   ↑ jne    67
```

KeyComp:

```
       0,92  │     add    %al,%bh
 0,02  │     sub    %al,(%rax)
 5,94  │     add    %al,(%rax)
 7,50  │     add    %al,%bh
       │     cmp    %al,(%rax)
 2,49  │     add    %al,(%rax)
 1,59  │     add    %al,-0x73(%rbx)
       │     rex.WR
 0,57  │     gs     (bad)
 0,09  │     shl    $0x2,%ecx
       │     mov    %ecx,%eax
13,86  │     and    $0x7,%eax
       │     lea    (%rcx,%rax,1),%edx
 2,94  │     mov    %edx,0x2c(%rbx)
 8,84  │     mov    %ebp,0x34(%rbx)
 1,84  │     movl   $0x0,0x3c(%rbx)
29,27  │     sub    $0x1c,%ecx
 0,58  │     mov    %ecx,0x30(%rbx)
10,36  │     movq   $0x0,0x20(%rbx)
 0,16  │     mov    %ebp,%eax
 0,14  │     mov    $0x0,%edx
       │     divl   0x2c(%rbx)
 0,59  │     mov    %eax,0x44(%rbx)
```

```
    │      mov    %eax,0x40(%rbx)
 0,46 │      mov    %r13d,0x48(%rbx)
 0,24 │      mov    %r12d,0x4c(%rbx)
11,28 │      mov    %r14,0x60(%rbx)
 0,31 │      .byte  0xbf
    │          ...
```

5.

```
Performance counter stats for './dc.B.x':

    480567,809306      task-clock (msec)         #    0,480 CPUs utilized
          234 710      context-switches          #    0,488 K/sec
            1 572      cpu-migrations            #    0,003 K/sec
          269 608      page-faults               #    0,561 K/sec
1 460 314 898 718      cycles                    #    3,039 GHz
1 784 271 081 861      instructions              #    1,22  insn per cycle
  453 039 940 525      branches                  #  942,718 M/sec
    7 384 700 319      branch-misses             #    1,63% of all branches
  444 155 290 550      L1-dcache-loads           #  924,230 M/sec
   30 456 594 482      L1-dcache-load-misses     #    6,86% of all L1-dcache hits
   16 304 253 726      LLC-loads                 #   33,927 M/sec
    8 528 686 254      LLC-load-misses           #   52,31% of all LL-cache hits

   1002,207608780 seconds time elapsed
```

Выводы

Для реализации поставленной задачи был взят NAS Parallel Benchmark DC.

Автоматическая оптимизация не дала большого ускорения, результаты замера показали замедление исполнения программы при использовании профилировщика.

В графе вызовов показывает, что наиболее затратные функции вставки узла в дерево и сравнение ключей. Также аннотированный листинг показал, что в данных функциях происходит наибольший процент кэш промахов, ошибок предсказания переходов.

Общая статистика показала, что наибольшая проблема данной программы кэш-промахи при обращении в память, что и является осовной временной затратой данной программы.