6.172 Project 1 Writeup

Albert Wang, Lekha Kuhananthan

September 26, 2011

1 2

1.1 2.1

We found that the memory leak was caused by not freeing the variable "right", so therefore we added $mem_free(\&right)$; to sort.c in the merge function, at the end and before the premature return statement. After this fix, valgrind reported no lost memory.

1.2 2.2

The perf annotated output for merge and sort showed that most of the program's time was spent in merge. Merge had the following output:

```
Merge:

Slowest in:

   17.12 /afs/csail.mit.edu/u/a/albertyw/project1/sort/sort.c:60

   16.79 /afs/csail.mit.edu/u/a/albertyw/project1/sort/sort.c:61

   13.90 /afs/csail.mit.edu/u/a/albertyw/project1/sort/sort.c:56

   13.79 /afs/csail.mit.edu/u/a/albertyw/project1/sort/sort.c:55

    5.51 /afs/csail.mit.edu/u/a/albertyw/project1/sort/sort.c:53
```

This means that most of the slowdown in the program is caused by branching and array indexing in the for loop of the merge function.

1.3 2.3

For an input of (2000, 100), the total time for sort without inline is 0.393289411 seconds elapsed while sort with inline is 0.393188160 seconds. This is a 0.03% speedup. The inline keyword is supposed to make the code speed up because inline causes the compiler to put the assembly code for a function in the place of where it is called, instead of making the assembly code jump to a different section for the function. This makes the compiled code slightly faster at the expense of using more memory.

However, the compiler can't do this everywhere.

Recursive functions can't do this because it would be impossible to place a function's code inside of itself. Also, inline code from a different file usually won't be inlined by the compiler because of dependencies.

1.4 2.4

The array indeces were changed to pointers, and for an input of (20000, 100), time reduced from 3.904939 seconds to 3.840331 seconds, a 1.65% speedup.

1.5 2.5

The original code had a perf stat output of

```
$ ./sort_p.64 10000000 10


 20277655203  branches                  #      0.000 M/sec
  1389280173  branch-misses             #      0.000 M/sec
 69447260155  cycles                    #      0.000 M/sec
 89566910390  instructions              #      1.290 IPC


26.059289767  seconds time elapsed
```

which means that the branch-misses to branches rate is 0.068512861. After modifying the code to remove the single if statement in the merge for loop using a bit hack, the new perf stat output was:

```
$ ./sort_p.64 10000000 10

 16757356921  branches                  #      0.000 M/sec
   208113220  branch-misses             #      0.000 M/sec
 62785649902  cycles                    #      0.000 M/sec
111701277873  instructions              #      1.779 IPC


23.560124343  seconds time elapsed
```

The new branch-misses to branches rate is 0.0124192151 and the program is 10% faster. Removing the branch requires additional instructions to compute the minimum.

1.6 2.6

Insertion sort is faster than merge sort for smaller inputs. Thus, one can perform a binary search through several input sizes to find the optimal size when insertion sort is a faster sort algorithm than merge sort. In our case, this transition is found at an input size of 100.

1.7 2.7

It is very easy to remove the `malloc` calls for `right`. Because of the way merge sort works, even if all the elements in `left` are smaller than the first element of `right`, the elements of `right` will never get overwritten even if they are left in the original array.

A compiler cannot automatically make these optimizations. It cannot predict that the space used by `right` will never be overwritten.

1.8 2.8

For an input of (20000, 100), the performance improved from 1.705721 seconds to 1.474061 seconds, a 13.5% speedup.

2 3

The general idea of the new rotate method is to split the rotation string into two parts, A and B. The parts are then reversed and switched using the identity $(a^R b^R)^R = ba$ This is currently implemented through two functions, bitarray_rotate and bitarray_reverse.

The bitarray_rotate function does some initial sanity checks, making sure

that the rotation is amount and bitarray length are nonzero. It then calls bitarray_reverse three times in order to do a bit shift as in the identity above. The bitarray_reverse method swaps individual bits in a section of the bitarray as called by bitarray_rotate.

The performance of the bitarray_rotate method should be O(n) where n is the length of the substring that is rotated. This contrasts with the original implementation which was O(m*n) where m is the number of bit shifts needed. More concretely, the execution time of the long-running rotation operation test was 0.001333s and the execution time of the long-running flip count operation test was 80.359986s. The original implementation had an execution time of 55.151890s for the former and 80.356707s for the latter. This is a 40,000 times increase in the speed of the rotation function.

There are several other methods that we will explore further. One plan is to use an inplace bit switch instead of using a temporary variable in the bitarray_reverse function. This should decrease running time because of less memory accesses and better caching. Another method is to have the bitarray_reverse function reverse bits in groups of 8 as bytes. This way, a byte reverse bithack can be used to considerably increase the function's speed. It is also expected that using pointers more would help with increasing bitar-

ray_rotate's speed.

The output for perf stat is below.

```
 Performance counter stats for './everybit -r':

      3.509498  task-clock-msecs         #      0.917 CPUs
             0  context-switches         #      0.000 M/sec
             0  CPU-migrations           #      0.000 M/sec
           195  page-faults              #      0.056 M/sec
       9343729  cycles                   #   2662.412 M/sec
      20357404  instructions             #      2.179 IPC
         22206  cache-references         #      6.327 M/sec
          1276  cache-misses             #      0.364 M/sec

    0.003825394  seconds time elapsed
```