

"Programming in the Large"

Design & Implement

• Program & programming style (done)

• Common data structures and algorithms (done)

• Modularity (done)

• Building techniques & tools (done)

Debug

• Debugging techniques & tools (done)

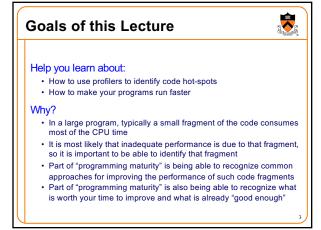
Test

• Testing techniques (done)

Maintain

• Performance improvement techniques & tools ← we are here

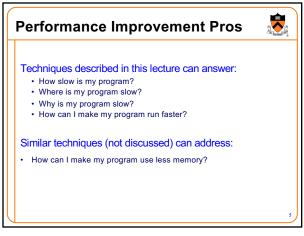
1



Agenda

Should you optimize?
What should you optimize?
Optimization techniques

3



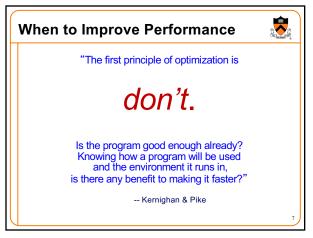
Performance Improvement Cons

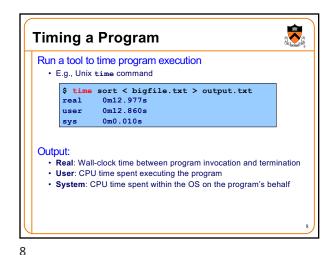
Techniques described in this lecture can yield code that:

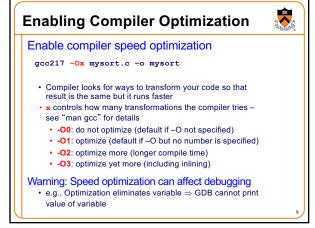
Is less clear/maintainable
Might confuse debuggers
Might contain bugs
Requires regression testing

So...

6



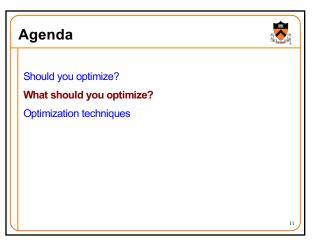




Now What?

So you've determined that your program is taking too long, even with compiler optimization enabled (and NDEBUG defined, etc.)

Is it time to rewrite the program?



Identifying Hot Spots

Spend time optimizing only the parts of the program that will make a difference!

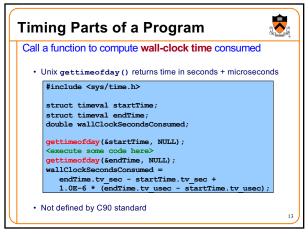
Gather statistics about your program's execution

• Coarse-grained: how much time did execution of a particular function call take?

• Time individual function calls or blocks of code

• Fine-grained: how many times was a particular function called? How much time was taken by all calls to that function?

• Use an execution profiler such as gprof



Timing Parts of a Program (cont.)

Call a function to compute CPU time consumed

• clock() returns CPU times in CLOCKS\_PER\_SEC units

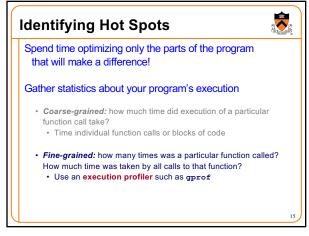
#include <time.h>
clock\_t startClock;
clock\_t endClock;
double cpuSecondsConsumed;

startClock = clock();
<execute some code here>
endClock = clock();
cpuSecondsConsumed =
 ((double) (endClock - startClock)) / CLOCKS\_PER\_SEC;

• Defined by C90 standard

14

16



GPROF Example Program

Example program for GPROF analysis

• Sort an array of 10 million random integers

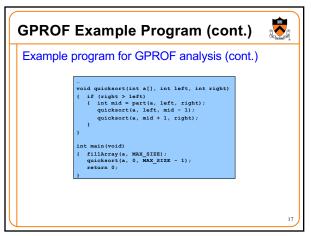
• Artificial: consumes lots of CPU time, generates no output

#include <atring.h>
#include <atrinc.h>
#include <atrinc.h

#include <atrinc.h>
#include <atrinc.h

#include <atrinc.h>
#include <atrinc.h

15



Using GPROF

Step 1: Instrument the program

gcc217 -pg mysort.c -o mysort

· Adds profiling code to mysort, that is...

· "Instruments" mysort

Step 2: Run the program

./mysort

· Creates file gmon.out containing statistics

Step 3: Create a report

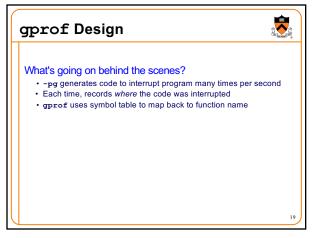
gprof mysort > myreport

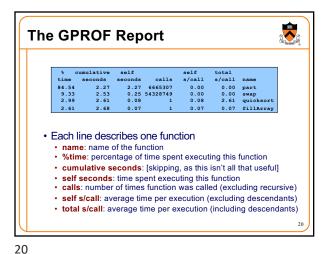
· Uses mysort and gmon.out to create textual report

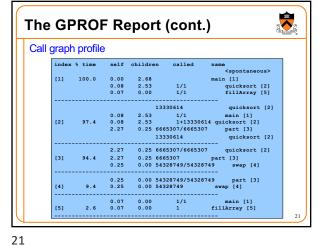
Step 4: Examine the report

cat myreport

17 18

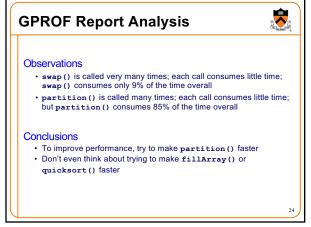




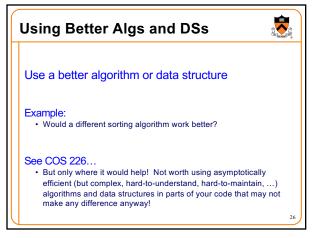


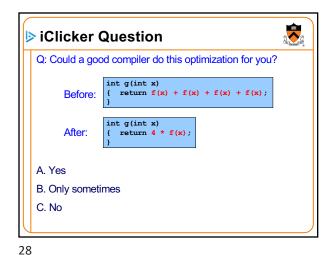
The GPROF Report (cont.) Call graph profile (cont.) · Each section describes one function · Which functions called it, and how much time was consumed? · Which functions it calls, how many times, and for how long? · Usually overkill; we won't look at this output in any detail

22









Aside: Side Effects as Blockers

int g(int x)
{ return f(x) + f(x) + f(x) + f(x);
}

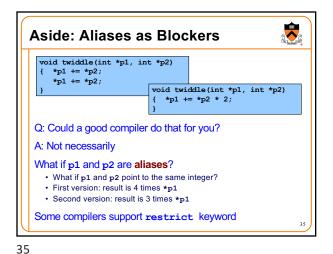
Q: Could a good compiler do that for you?

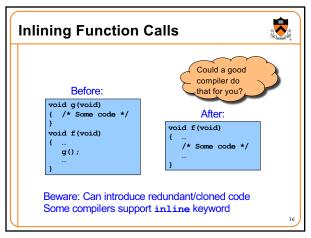
A: Only sometimes...

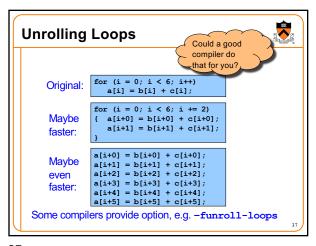
Suppose f() has side effects?

int counter = 0;
...
int f(int x)
{ return counter++;
}

And f() might be defined in another file known only at link time!







Rewrite code in a lower-level language

• As described in this module of the course ...

• Compose key functions in assembly language instead of C

• Use registers instead of memory

• Use instructions (e.g. adc) that compiler doesn't know

Beware: Modern optimizing compilers generate fast code

• Hand-written assembly language code could be slower!

