High Performance Computing
Lecture 33

Matthew Jacob

Indian Institute of Science

Profiling

- Profiler: A tool that helps you identify the `important' parts of your program to concentrate your optimization efforts
- Profile: a breakup (of execution time) across the different parts of the program
- Can be done by adding statements to your program (instrumentation) -- so that during execution, data is gathered, outputted and possibly processed later
- Automation: where a profiling tool adds those instructions into your program for you

Profiling Mechanisms

- Levels of Granularity typically supported
  - Function level
  - Statement level
  - Basic block level: A basic block is a sequence of contiguous instructions in a program with a single entry point (the first instruction in the basic block) and a single exit point (the last instruction in the basic block)
- Two examples of profile data
  - execution time
  - execution counts
- We will look at examples of profiling mechanisms at the function and basic block level

Why Function Level Profiling?

- How useful can it be to identify and optimize a few functions of a program?
- Example: LINPACK Benchmark
  - LINPACK: A Linear Algebra package
  - The benchmark solves a large system of linear equations by Gaussian elimination using LINPACK routines
    - Benchmark programs are used to compare the performance of computer systems
  - It spends most (~70%) of its run time in SAXPY

Prof: UNIX Function Level Profiling

- Usage

  % cc –p program.c /generates instrumented a.out

  % a.out / execution; instrumentation

  / generates data and mon.out

  % prof / processing of profile data

- Output gives a function by function breakup of execution time
- Useful in identifying which functions to concentrate optimization efforts on

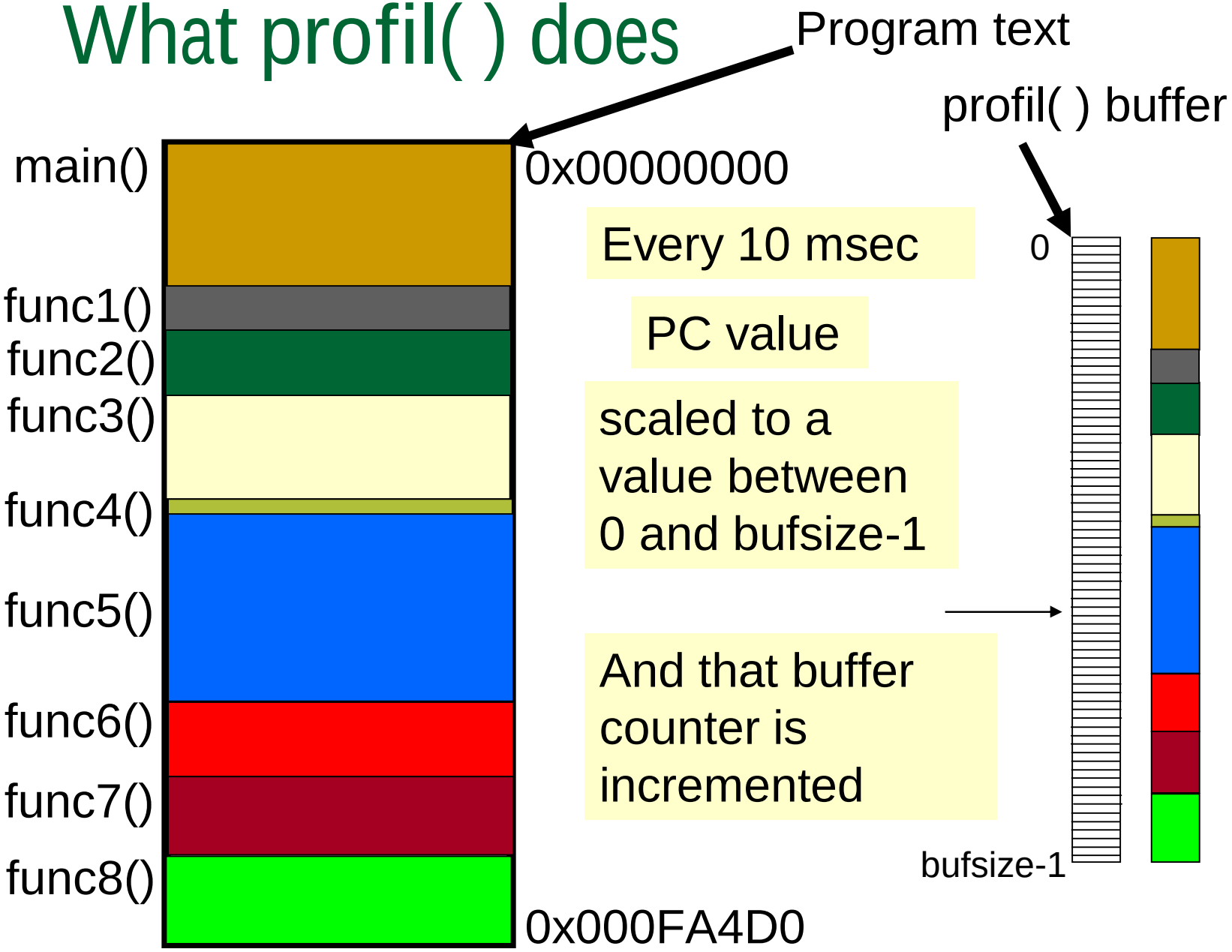
How prof works

- The instrumentation does three things
  1. At entry of each function: increment an execution count for that function
  2. At program entry: make a call to system call **profil()** to get execution times in each function
  3. At program exit: write profile data to an output file that can later be processed
- profil(): execution time profiler
  - Generates a histogram of the execution time in each function

What profil( ) does

- One of the parameters in call to profil() is a buffer
- It is used as an array of counters initialized to 0
- The array elements are associated with contiguous regions of the program text
- During program execution
  - PC value is sampled
  - once every clock tick (typical default: 10 msec)
  - triggered by the hardware timer interrupt
- Corresponding buffer element is incremented

What profil( ) does



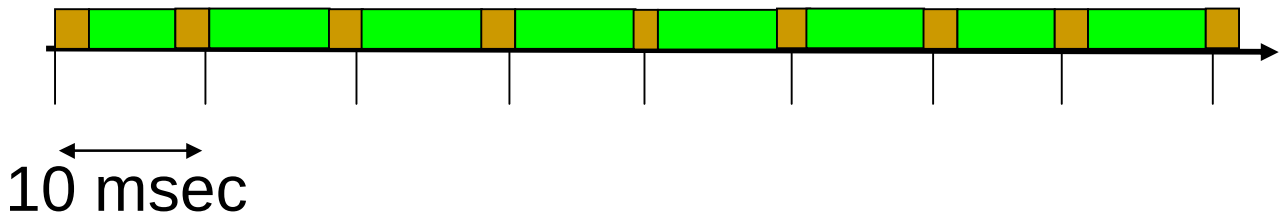
Output: Matrix Multiply

%Time	Seconds	CumSecs	#Calls	Name
91.0	11.79	11.79	10000	_write
5.4	0.70	12.49	10000	MultStep
1.2	0.16	12.65	10000	_doprnt
0.9	0.12	12.77		_mcount
0.4	0.05	12.82	10000	printf
0.2	0.02	12.84	1	ReadIn
...				
0.0	0.00	12.96	1	main
0.0	0.00	12.96	1	PrintOut
0.0	0.00	12.96	1	Multiply

Using prof

- From how it works, we understand that
  - The granularity is at best 10 msec
  - The generated profile could differ for multiple runs of a program running on the same input data
    - Remember that there could be other programs running on the same system
    - This can affect the behaviour of the profiling run in terms of page faults, cache misses, etc
  - And could even be completely wrong
    - e.g., there could be a particular function that just happens to be running each time the timer interrupt occurs

Prof giving bad time estimates



A( )

B( )

The prof profile will show 100% of the execution time being spent in function A()

Using prof

- Some usage guidelines
  - Do the profile run under light load conditions
  - Do the profiling run a few times and see if the results vary a lot
  - Remember that the function execution counts are exact, even though the execution times are only estimates

Pixie: Basic Block Level Profiling

- A different style of profiling
- Usage

```
% cc program.c          / a.out
% pixie a.out           / instrumented a.out.pixie
% a.out.pixie           / profile output file
% prof                  / report on profile data
```

- Output is based on basic block level execution counts
- Useful for all kinds of things

What is a Basic Block?

- A section of program that does not cross any conditional branches, loop boundaries or other transfers of control
- A sequence of instructions with a single entry point, single exit point, and no internal branches
- A sequence of program statements that contains no labels and no branches
- A basic block can only be executed completely and in sequence

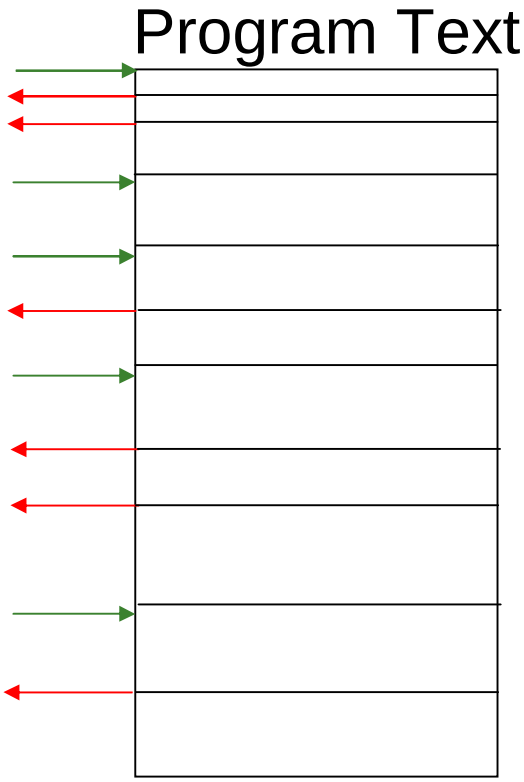
Pixie: How it works

1. Identification of basic blocks
   - Q: How can basic blocks be identified?
   - Pixie uses heuristics where necessary

2. Instrumentation

Identifying Basic Blocks of a Program

- Basic blocks are defined by control transfer instructions and their targets



Problem case:

JR R8

- Target address known only when program runs
- Target address can be different each time instruction is executed

Pixie: How it works

1. Identification of basic blocks
   - Q: How can basic blocks be identified?
   - Pixie uses heuristics where necessary

2. Instrumentation
   - Increment a counter for the basic block
   - On program entry and exit: initialization of data structures; writing profile output file

How intrusive are these mechanisms?

- Issue: Does the instrumented program behave enough like the original program?
  - If not, the profile generated might mislead the direction of program optimization efforts

How intrusive are these mechanisms?

- Pixie
  - The instrumented executable program can be much larger than the original program

Pixie instrumentation

- In each basic block, instructions must be added to increment an execution counter for that basic block
  - The counters cannot be maintained in registers
    - There can be a lot of basic blocks in a program
  - At least three MIPS 1 instructions would be needed
    - LW R1, counter
    - ADDI R1, R1, 1
    - SW counter, R1

Pixie instrumentation

- How big is the typical basic block?
- How frequent are control transfer instructions?
  - around 20% of all instructions executed
  - So, average basic block size might be about 5 instructions
  - to which 3 instructions must be added to increment the basic block execution count

How intrusive are these mechanisms?

- Pixie
  - The instrumented executable program can be much larger than the original program
  - Does not matter; basic block execution counts are accurate
- Prof: gathers more than just execution counts
  - Instrumentation is not very large

Other Profiling Tools

- Intel® VTune™ Performance Analyzer
  - Available for Windows and also for Linux
  - Provides a way to access hardware performance counters
    - Hardware counting mechanism like the timestamp counter
    - A variety of hardware events can be counted during program execution
    - Examples: Instructions executed, Cache misses, Branch mispredictions