Code Profiling

CSE260, Computer Science B: Honors

Stony Brook University

http://www.cs.stonybrook.edu/~cse260

Performance

- Programs should:
  - solve a problem correctly
  - be readable
  - be flexible (for future modifications)
  - ***be fast***
  - ***be lean (use the least amount of memory necessary)***
- Hardware becomes faster and more capable, but data is more.
  - program efficiency is still an issue
  - software expectations continue to increase

When not to optimize

- Make sure you only optimize when necessary
- Why not just be responsible and always optimize?
  - optimization is a good way to introduce bugs ☺
  - some techniques decrease the portability of your code
  - you can expend a lot of effort for little or no results
  - optimization makes the code less readable!
    - Increases complexity and the difficulty of debugging.
  - optimization can be hard to do properly
    - optimization is a moving target
- Donald Knuth famously said:

"*Premature optimization is the root of all evil.*"

So what is optimizing?

- Simply, the process of modifying a system to improve it's performance
- Why does Donald Knuth hate it?
  - because all the optimization in the world can never replace the wise selection of proper data structures and the proper use of efficient algorithms
  - Donald Knuth is credited with creating the rigorous academic discipline of algorithm analysis
  - John Carmack might say, because he never made Doom with tight hardware constraints: optimization is commonly done during graphics-intensive game development because few other programs have the performance requirements that games do
    - He had to make use of several tricks for these features to run smoothly on home computers of 1993: the levels were not truly three-dimensional (they were internally represented on a single plane, with height differences stored separately as displacements.

Optimization Options

- What options are available?
  - ***Design for Speed*** - select your data structures & algorithms wisely!
  - Use a code profiler
  - Use automatic optimizers
  - Implement low-level code optimizations

Design for Speed

- In the process of designing a program:
  - Identify operations needed in each class before choosing your data structures.
  - Pick a data structure that can be used to perform such operations efficiently.
  - Select a solid algorithm that fits the problem and the data:
    - Big O notation.
  - It may not be always possible to know what is the best choice:
    - Choice may depend on amount of data and the frequency of its use.
    - Choices may trade off readability and maintainability for speed or memory performance.

Program Hot Spots

"90% of the time is spent in 10% of the code."

- When the choice of "best" data structure or algorithm is not clear at design time:
  - use the simplest data structure or algorithm,
  - collect data about the impact of the data structure or algorithm on the overall speed of the program.
- Identify the portion of the code that takes the most time or memory.
- Replace that section, if possible, with a better data structure or algorithm
  - based on the "frequency of use" data you have collected.

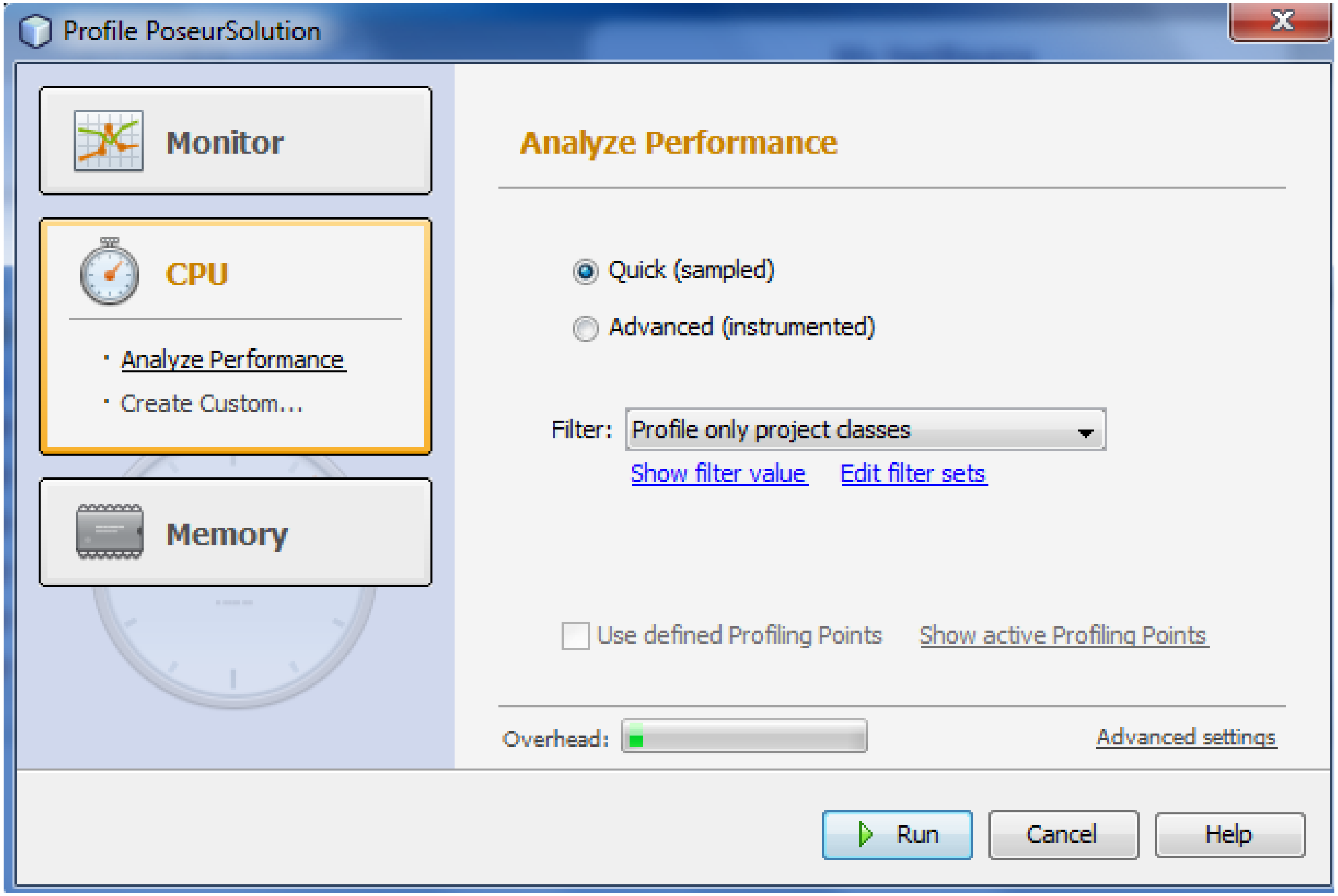
What’s a profiler?

- A profiler is a program that can track the performance of another program by checking information collected while the code is executing:
  - can usually track time used or frequency of use of code portions (random sampling with a parallel thread),
  - the entire application or just select methods.
- Types of profiling:
  - CPU performance profiling,
  - Memory profiling,
  - Threads profiling,
  - Memory leak profiling.

Java Profilers & Optimizers

- Eclipse
  - Eclipse Test and Performance Tools Platform Project (TPTP)
    - http://www.eclipse.org/tptp/index.php
  - Eclipse Colourer, by Konstantin Scheglov – a free plugin from sourceforge.net
    - http://sourceforge.net/projects/eclipsecolorer
    - http://www.theserverside.com/news/1364402/Code-Analysis-with-the-Eclipse-Profiler
- NetBeans
  - http://profiler.netbeans.org - uses JFluid technology
- Borland's Optimizeit (for Java, &.NET, etc …)
    - http://techpubs.borland.com/optimizeit/index.html
- JProbe by Quest Software (acquired by Dell in 2012)
  - http://www.quest.com

NetBeans Profiler Setup



Profile PoseurSolution

- Analyze Performance
- Create Custom...

Analyze Performance

[✓ Quick (sampled)]
Advanced (instrumented)

Filter: Profile only project classes

☐ Use defined Profiling Points

Overhead:

Collecting Performance Data using HProf

- Generate profile data on sample runs:
  - `java -classic -Xrunhprof Driver`
  - `java -classic -Xrunhprof:cpu=samples Driver`
- Analyze the profile data to find:
  - Hot spots with respect to time
    - Most frequently used methods
    - Most time-consuming methods
      - Bottlenecks
  - Hot spots with respect to space (memory)
    - Most frequently used portions of data structures

Profiling Example 1

```
import java.util.*;
public class TestHprof1 {
  public static ArrayList sortArrayList(){
      ArrayList list = new ArrayList();
      int num;
      for (int i = 0; i < 20000; i++) {
           num = (int)(Math.random() * 100000);
           list.add(new Integer(num));
      }
      Collections.sort((List<Integer>) list);
      return list;
  }
```

```
    public static Vector sortVector() {
        Vector v = new Vector();
        int num;
        for (int i = 0; i < 20000; i++) {
            num = (int)(Math.random() * 100000);
            v.add(new Integer(num));
        }
        Collections.sort(v);
        return v;
    }
    public static void main(String[] args) {
        ArrayList list = sortArrayList();
        System.out.println(list.size());
        Vector v = sortVector();
        System.out.println(v.size());
    }
}
```

http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

Output of hprof

```
>java -classic -Xrunhprof:cpu=samples TestHprof1
Warning: classic VM not supported; client VM will be used
20000
20000
Dumping Java heap ... allocation sites ... done.
 Created 2 files:
   java.hprof.txt.TMP
   java.hprof.txt
CPU SAMPLES BEGIN (total = 13)
rank self   accum count trace  method
1   38.46% 38.46%   5   8  TestHprof.sortVector
2   23.08% 61.54%   3   4  TestHprof.sortArrayList
3    7.69% 69.23%   1   7 CollectionsComparison.main
4    7.69% 76.92%   1   2 java.lang.ClassLoader.checkCerts
5    7.69% 84.62%   1   6 java.util.AbstractList.listIterator
6    7.69% 92.31%   1   5 java.util.Arrays.mergeSort
7    6.67% 100.00%  1   4 sun.misc.AtomicLongCSImpl.attemptUpdate
CPU SAMPLES END
(remaining entries less than 1% each, omitted for brevity)
```

http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

Output of hprof

1. Execution of a Java program typically involves many library calls, making it difficult to gather useful information from profiles.

2. Call graph information (frequency of calls) is not given explicitly by hprof or other Java profilers.

Additional Optimization Tools

- Compiler optimization
  - Compilers already perform many optimizations
    - Example: dead code elimination
    - http://www.javaworld.com/javaworld/jw-03-2000/jw-03-javaperf_4.html
  - can't be used while debugging
- Just-In-Time compilers
  - compiles only code necessary at runtime
- Optimizer tools (many times confused with profilers)
  - Ex: jarg by SourceForge – reduces the size of a jar file in which java class files are stored
    - http://jarg.sourceforge.net/
  - **javac –O, the Java optimizer**
    - Optimizes compiled code by inlining static, final and private methods.
- Assembly code optimization?

Inlining using javac –O

- The Java optimizer works by inlining selected methods
- Inlining a method call inserts the code for the method directly into the code making the method call
  - Eliminates the overhead of the method call
  - For a small method this overhead can represent a significant percentage of its execution time
  - Only private, static, or final methods are eligible for inlining
  - Synchronized methods won't be inlined
  - The compiler will only inline small methods typically consisting of only one or two lines of code

Machine Code

- javac compiles Java programs to the "machine code" for the Java Virtual Machine (JVM)
  - directly calling this code can improve program performance
- javap –c to dump out the JVM instructions generated by the Java compiler (see if hand-optimization is necessary)
  - gcc –S to dump assembly instructions generated by the C compiler

javap Example: TestHprof1

```
> javap -c TestHprof1
Compiled from "TestHprof1.java"
public class TestHprof1 {
  public TestHprof1();
    Code:
       0: aload_0
       1: invokespecial #1  // Method java/lang/Object."<init>":()V
       4: return
public static void main(java.lang.String[]);
    Code:
       0: invokestatic  #14   // Method sortArrayList:()Ljava/util/ArrayList;
       3: astore_1
       4: getstatic     #15   // Field java/lang/System.out:Ljava/io/PrintStream;
       7: aload_1
       8: invokevirtual #16   // Method java/util/ArrayList.size:()I
      11: invokevirtual #17   // Method java/io/PrintStream.println:(I)V
      14: invokestatic  #18   // Method sortVector:()Ljava/util/Vector;
      17: astore_2
      18: getstatic     #15   // Field java/lang/System.out:Ljava/io/PrintStream;
      21: aload_2
      22: invokevirtual #19   // Method java/util/Vector.size:()I
      25: invokevirtual #17   // Method java/io/PrintStream.println:(I)V
      28: return
```

```
public static java.util.ArrayList sortArrayList();
  Code:
   0:   new     #2; //class ArrayList
   3:   dup
   4:   invokespecial   #3; //Method java/util/ArrayList."<init>":()V
   7:   astore_0
   8:   iconst_0
   9:   istore_2
   10:  iload_2
   11:  sipush  20000
   14:  if_icmpge       45
   17:  invokestatic    #4; //Method java/lang/Math.random:()D
   20:  ldc2_w  #5; //double 100000.0d
   23:  dmul
   24:  d2i
   25:  istore_1
   26:  aload_0
   27:  new     #7; //class Integer
   30:  dup
   31:  iload_1
   32:  invokespecial   #8; //Method java/lang/Integer."<init>":(I)V
   35:  invokevirtual   #9; //Method java/util/ArrayList.add:(Ljava/lang/Object;)Z
   38:  pop
   39:  iinc    2, 1
   42:  goto    10
   45:  aload_0
   46:  invokestatic    #10; //Method java/util/Collections.sort:(Ljava/util/List;)V
   49:  aload_0
   50:  areturn
```

```
public static java.util.Vector sortVector();
  Code:
   0:   new     #11; //class Vector
   3:   dup
   4:   invokespecial   #12; //Method java/util/Vector."<init>":()V
   7:   astore_0
   8:   iconst_0
   9:   istore_2
   10:  iload_2
   11:  sipush  20000
   14:  if_icmpge       45
   17:  invokestatic    #4; //Method java/lang/Math.random:()D
   20:  ldc2_w  #5; //double 100000.0d
   23:  dmul
   24:  d2i
   25:  istore_1
   26:  aload_0
   27:  new     #7; //class Integer
   30:  dup
   31:  iload_1
   32:  invokespecial   #8; //Method java/lang/Integer."<init>":(I)V
   35:  invokevirtual   #13; //Method java/util/Vector.add:(Ljava/lang/Object;)Z
   38:  pop
   39:  iinc    2, 1
   42:  goto    10
   45:  aload_0
   46:  invokestatic    #10; //Method java/util/Collections.sort:(Ljava/util/List;)V
   49:  aload_0
   50:  areturn
```

JNI

- JNI – Java Native language Interface
  - Standard programming interface for writing Java native methods and embedding the Java^{TM} virtual machine into native applications.
  - http://docs.oracle.com/javase/1.5.0/docs/guide/jni/spec/jniTOC.html
  - used on really critical components.
  - trade portability for performance
  - also allows C/C++ programs to execute Java code
    - JNI enables one to write native methods to handle situations when an application cannot be written entirely in the Java programming language, e.g. when the standard Java class library does not support the platform-specific features or program library.
    - C code will also run faster than Java
    - Some coders will use JNI to run bottlenecked parts of their programs in C

It is like using Assembly Code in C++

- Example method for a 32-bit memory fill funtion:

```
void qmemset(void *memory, int value, int num_quads){
_asm
{ CLD                          // clear the direction flag
  MOV EDI, memory              // move pointer into EDI
  MOV ECX, num_quads// ECX hold loop count
  MOV EAX, value               // EAX hold value
  REP STOSD                    // perform fill
}
}
…
qmemset(&buffer, 25, 1000);
```

- Inlining assembly should only be done if you can write assembly code better than the compiler can generate it

Look-up Tables

- Pre-computed values of some computation that you know you'll perform during run-time
- Simply compute all possible values at startup then run the game
  - Common game look-up:
    - pre-compute sin & cos of all angles 0 – 359 degrees
    - place values in a simple array
    - faster to look-up array values than perform operation

Fibonacci Numbers

```
Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89…
         indices: 0 1 2 3 4 5 6 7  8  9  10 11
```

fib(0) = 0;

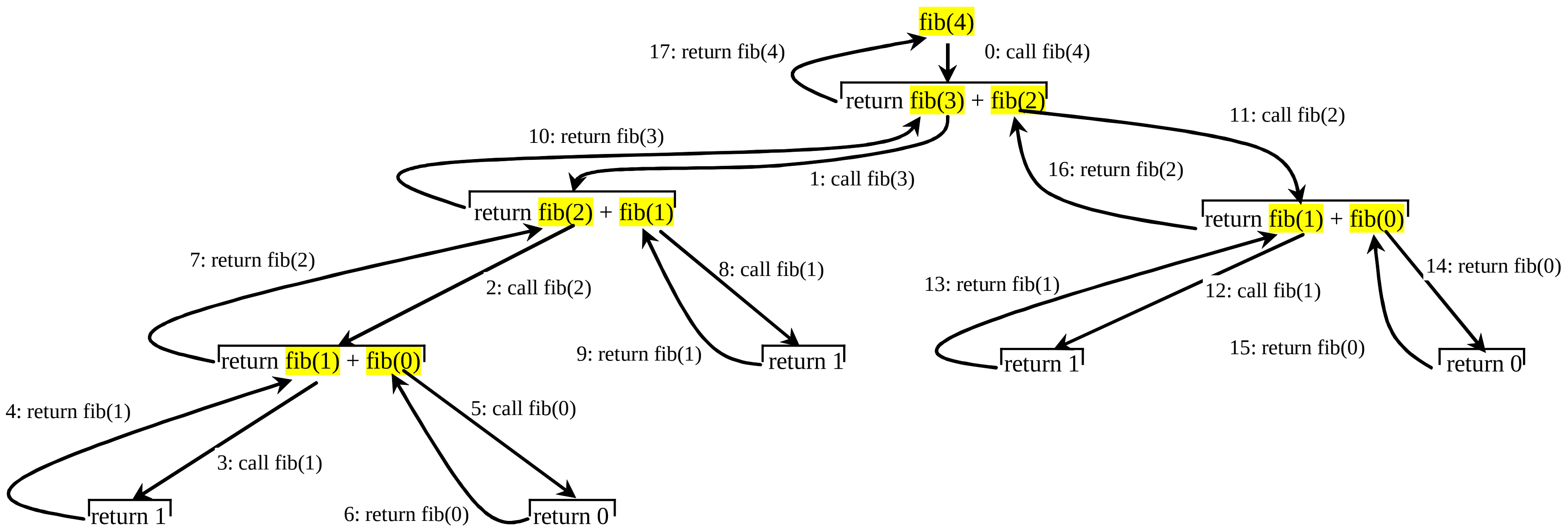
fib(1) = 1;

fib(index) = fib(index -1) + fib(index -2);   for integers index >=2

$$\boxed{\begin{aligned}\text{fib}(3) &= \text{fib}(2) + \text{fib}(1) = (\text{fib}(1) + \text{fib}(0)) + \text{fib}(1)\\ &= (1 + 0) + \text{fib}(1) = 1 + \text{fib}(1) = 1 + 1 = 2\end{aligned}}$$

```
import java.util.Scanner;
public class ComputeFibonacci {
public static void main(String args[]) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter an index for the Fibonacci number: ");
    int index = input.nextInt();
    // Find and display the Fibonacci number
    System.out.println("Fibonacci(" + index + ") is " + fib(index));
  }
  /** The method for finding the Fibonacci number */
  public static long fib(long index) {
    if (index == 0) // Base case
      return 0;
    else if (index == 1) // Base case
      return 1;
    else  // Reduction and recursive calls
      return fib(index - 1) + fib(index - 2);
  }
}
```

Fibonnaci Numbers



```
import java.util.Scanner;
public class ComputeFibonacciTabling {  // NO REPEATED COMPUTATION
public static void main(String args[]) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter an index for the Fibonacci number: ");
    int index = input.nextInt();
    f = new long[n];
    System.out.println("Fibonacci(" + index + ") is " + fib(index));
  }
  public static long[] f;
  public static long fib(long index) {
    if (index == 0)       return 0;
    if (index == 1) {      f[1]=1;      return 1;   }
    if(f[index]!=0)
      return f[index];
    else  // Reduction and recursive calls
      f[index] = fib(index - 1) + fib(index - 2);
    return f[index];
  }
}
```

Optimization Guidelines

- Do NOT hand-optimize your code:
  - if it unnecessarily sacrifices readability, OR
  - if it unnecessarily sacrifices maintainability
- If an optimization is necessary, think data structures & algorithms first
- Again, many common code optimizations are done by the compiler.
  - If it becomes necessary, find out what optimizations are done automatically by the compiler you are using
  - Compilers get smarter all the time…
- If you over-optimize, your fellow coders will hate you